

# Underapproximation of Procedure Summaries for Integer Programs

Pierre Ganty<sup>1</sup>, Radu Iosif<sup>2</sup>, and Filip Konecny<sup>2</sup>

<sup>1</sup> IMDEA Software Institute, Madrid, Spain

<sup>2</sup> VERIMAG/CNRS, Grenoble, France

**Abstract.** We show how to underapproximate the procedure summaries of recursive programs over the integers using off-the-shelf analyzers for non-recursive programs. The novelty of our approach is that the non-recursive program we compute may capture unboundedly many behaviors of the original recursive program for which stack usage cannot be bounded. Moreover, we identify a class of recursive programs on which our method terminates and returns the precise summary relations without underapproximation. Doing so, we generalize a similar result for non-recursive programs to the recursive case. Finally, we present experimental results of an implementation of our method applied on a number of examples.

## 1 Introduction

Procedure summaries are relations between the input and return values of a procedure, resulting from its terminating executions. Computing summaries is important, as they are a key enabler for the development of modular verification techniques for interprocedural programs, such as checking safety, termination or equivalence properties. Summary computation is, however, challenging in the presence of *recursive procedures* with integer parameters, return values, and local variables. While many analysis tools exist for non-recursive programs, only a few ones address the problem of recursion.

In this paper, we propose a novel technique to generate arbitrarily precise *underapproximations* of summary relations. Our technique is based on the following idea. The control flow of procedural programs is captured precisely by the language of a context-free grammar. A  $k$ -index underapproximation of this language (where  $k \geq 1$ ) is obtained by filtering out those derivations of the grammar that exceed a budget, called *index*, on the number (at most  $k$ ) of occurrences of non-terminals occurring at each derivation step. As expected, the higher the index, the more complete the coverage of the underapproximation. From there we define the  $k$ -index summary relations of a program by considering the  $k$ -index underapproximation of its control flow.

Our method then reduces the computation of  $k$ -index summary relations for a recursive program to the computation of summary relations for a non-recursive program, which is, in general, easier to compute because of the absence of recursion. The reduction was inspired by a decidability proof [4] in the context of Petri nets.

The contributions of this paper are threefold. First, we show that, for a given index, recursive programs can be analyzed using off-the-shelf analyzers designed for non-recursive programs. Second, we identify a class of recursive programs, with possibly

unbounded stack usage, on which our technique is complete i.e., it terminates and returns the precise result. Third, we present experimental results of an implementation of our method applied on a number of examples.

**Related Work** The problem of analyzing recursive programs handling integers (in general, unbounded data domains) has gained significant interest with the seminal work of Sharir and Pnueli [24]. They proposed two approaches for interprocedural dataflow analysis. The first one keeps precise values (*call strings*) up to a limited depth of the recursion stack. In contrast to the methods based on the call strings approach, our method can also analyse precisely certain programs for which the stack is unbounded.

The second approach of Sharir and Pnueli is based on computing the least fixed point of a system of recursive dataflow equations (the *functional approach*). This approach to interprocedural analysis is based on computing an increasing *Kleene sequence* of summaries for control paths in the program of increasing, but *bounded length*. Recently [11], the *Newton sequence* was shown to converge at least as fast as the Kleene sequence. The intuition behind the Newton sequence is to consider control paths in the program of increasing *index*, and *unbounded length*. Our contribution can be seen as a technique to compute the iterates of the Newton sequence for programs with integer parameters, return values, and local variables.

The complexity of the functional approach was shown to be polynomial in the size of the (finite) abstract domain, in the work of Reps, Horwitz and Sagiv [23]. This result is achieved by computing summary information, in order to reuse previously computed information during the analysis. Following up on this line of work, most existing abstract analyzers, such as INTERPROC [19], also use relational domains to compute *overapproximations* of function summaries – typically widening operators are used to ensure termination of fixed point computations. The main difference of our method with respect to static analyses is the use of underapproximation instead of overapproximation. If the final purpose of the analysis is program verification, our method will not return false positives. Moreover, the coverage can be increased by increasing the bound on the derivation index.

Previous works have applied model checking based on abstraction refinement to recursive programs. One such method, known as *nested interpolants* represents programs as nested word automata [3], which have the same expressive power as the visibly push-down grammars used in our paper. Also based on interpolation is the WHALE algorithm [2], which combines partial exploration of the execution paths (underapproximation) with the overapproximation provided by a predicate-based abstract post operator, in order to compute summaries that are sufficient to prove a given safety property. Another technique, similar to WHALE, although not handling recursion, is the SMASH algorithm [15] which combines may- and must-summaries for compositional verification of safety properties. These approaches are, however, different in spirit from ours, as their goal is proving given safety properties of programs, as opposed to computing the summaries of procedures independently of their calling context, which is our case. We argue that summary computation can be applied beyond safety checking, e.g., to prove termination [5], or program equivalence.

## 2 Preliminaries

**Grammars** A *context-free grammar* (or simply grammar) is a tuple  $G = (X, \Sigma, \delta)$  where  $X$  is a finite nonempty set of *nonterminals*,  $\Sigma$  is a finite nonempty *alphabet* and  $\delta \subseteq X \times (\Sigma \cup X)^*$  is a finite set of *productions*. The production  $(X, w)$  may also be noted  $X \rightarrow w$ . Also define  $\text{head}(X \rightarrow w) = X$  and  $\text{tail}(X \rightarrow w) = w$ . Given two strings  $u, v \in (\Sigma \cup X)^*$  we define a *step*  $u \Rightarrow v$  if there exists a production  $(X, w) \in \delta$  and some words  $y, z \in (\Sigma \cup X)^*$  such that  $u = yXz$  and  $v = ywz$ . We use  $\Rightarrow^*$  to denote the reflexive transitive closure of  $\Rightarrow$ . The *language* of  $G$  produced by a nonterminal  $X \in X$  is the set  $L_X(G) = \{w \in \Sigma^* \mid X \Rightarrow^* w\}$  and we call any sequence of steps from a nonterminal  $X$  to  $w \in \Sigma^*$  a *derivation* from  $X$ . Given  $X \Rightarrow^* w$ , we call the sequence  $\gamma \in \delta^*$  of productions used in the derivation a *control word* and write  $X \xRightarrow{\gamma} w$  to denote that the derivation conforms to  $\gamma$ .

**Visibly Pushdown Grammars** To model the control flow of procedural programs we use languages generated by visibly pushdown grammars, a subset of context-free grammars. In this setting, words are defined over a *tagged alphabet*  $\hat{\Sigma} = \Sigma \cup \langle \Sigma \cup \Sigma \rangle$ , where  $\langle \Sigma = \{\langle a \mid a \in \Sigma \rangle\}$  intuitively represents procedure call site and  $\Sigma \rangle = \{a \rangle \mid a \in \Sigma\}$  represents procedure return site. Formally, a *visibly pushdown grammar*  $G = (X, \hat{\Sigma}, \delta)$  is a grammar that has only productions of the following forms, for some  $a, b \in \Sigma$ :

$$X \rightarrow a \qquad X \rightarrow aY \qquad X \rightarrow \langle aYb \rangle Z$$

It is worth pointing that, for our purposes, we do not need a visibly pushdown grammar to generate the empty string  $\epsilon$ . Each tagged word generated by visibly pushdown grammars is associated a *nested word* [3] the definition of which we briefly recall. Given a finite alphabet  $\Sigma$ , a *nested word* over  $\Sigma$  is a pair  $(w, \rightsquigarrow)$ , where  $w = a_1a_2 \dots a_n \in \Sigma^*$ , and  $\rightsquigarrow \subseteq \{1, 2, \dots, n\} \times \{1, 2, \dots, n\}$  is a set of *nesting edges* (or simply edges) where:

1.  $i \rightsquigarrow j$  only if  $i < j$ , i.e. edges only go forward;
2.  $|\{j \mid i \rightsquigarrow j\}| \leq 1$  and  $|\{i \mid i \rightsquigarrow j\}| \leq 1$ , i.e. no two edges share a position;
3. if  $i \rightsquigarrow j$  and  $k \rightsquigarrow \ell$  then it is not the case that  $i < k \leq j < \ell$ , i.e. edges do not cross.

Intuitively, we associate a nested word to a tagged word as follows: there is an edge between tagged symbols  $\langle a$  and  $b \rangle$  iff both are generated at the same derivation step. For instance looking forward at Ex. 2 consider the tagged word  $w = \tau_1\tau_2\langle\tau_3\tau_1\tau_5\tau_6\tau_7\tau_3\rangle\tau_4$  resulting from a derivation  $Q_1^{\text{init}} \Rightarrow^* w$ . The nested word associated to  $w$  is  $(\tau_1\tau_2\tau_3\tau_1\tau_5\tau_6\tau_7\tau_3\tau_4, \{3 \rightsquigarrow 8\})$ . Finally, let  $w \mapsto n w$  denote the mapping which given a tagged word in the language of a visibly pushdown grammar returns the nested word thereof.

**Integer Relations** We denote by  $\mathbb{Z}$  the set of integers. Let  $\mathbf{x} = \{x_1, x_2, \dots, x_d\}$  be a set of variables for some  $d > 0$ . Define  $\mathbf{x}'$  the *primed* variables of  $\mathbf{x}$  to be  $\{x'_1, x'_2, \dots, x'_d\}$ . All variables range over  $\mathbb{Z}$ . We denote by  $\overrightarrow{\mathbf{y}}$  an ordered sequence  $\langle y_1, \dots, y_k \rangle$  of variables, and by  $|\overrightarrow{\mathbf{y}}|$  its length  $k$ . By writing  $\overrightarrow{\mathbf{y}} \subseteq \mathbf{x}$  we mean that each variable in  $\overrightarrow{\mathbf{y}}$  belongs to  $\mathbf{x}$ . For sequences  $\overrightarrow{\mathbf{y}}$  and  $\overrightarrow{\mathbf{z}}$  of length  $k$ , let  $\overrightarrow{\mathbf{y}} = \overrightarrow{\mathbf{z}}$  stand for the equality  $\bigwedge_{i=1}^k y_i = z_i$ .

A *linear term*  $t$  is a linear combination of the form  $a_0 + \sum_{i=1}^d a_i x_i$ , where  $a_0, a_1, \dots, a_d \in \mathbb{Z}$ . An *atomic proposition* is a predicate of the form  $t \leq 0$ , where  $t$  is a linear term. We consider formulae in the first-order logic over atomic propositions  $t \leq 0$ , also known as *Presburger arithmetic*.

A *valuation* of  $\mathbf{x}$  is a function  $v : \mathbf{x} \rightarrow \mathbb{Z}$ . The set of all valuations of  $\mathbf{x}$  is denoted by  $\mathbb{Z}^{\mathbf{x}}$ . If  $\vec{\mathbf{y}} = \langle y_1, \dots, y_k \rangle$  is an ordered sequence of variables, we denote by  $v(\vec{\mathbf{y}})$  the sequence of integers  $\langle v(y_1), \dots, v(y_k) \rangle$ . An arithmetic formula  $\mathcal{R}(\mathbf{x}, \mathbf{y}')$  defining a respect to two valuations  $v_1 \in \mathbb{Z}^{\mathbf{x}}$  and  $v_2 \in \mathbb{Z}^{\mathbf{y}'}$ , by replacing each  $x \in \mathbf{x}$  by  $v_1(x)$  and each  $y' \in \mathbf{y}'$  by  $v_2(y')$  in  $\mathcal{R}$ . The composition of two relations  $R_1 \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{y}}$  and  $R_2 \subseteq \mathbb{Z}^{\mathbf{y}} \times \mathbb{Z}^{\mathbf{z}}$  is denoted by  $R_1 \circ R_2 = \{(\mathbf{u}, \mathbf{v}) \in \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{z}} \mid \exists \mathbf{t} \in \mathbb{Z}^{\mathbf{y}}. (\mathbf{u}, \mathbf{t}) \in R_1 \text{ and } (\mathbf{t}, \mathbf{v}) \in R_2\}$ . For a subset  $\mathbf{y} \subseteq \mathbf{x}$ , we denote  $v \downarrow_{\mathbf{y}} \in \mathbb{Z}^{\mathbf{y}}$  the projection of  $v$  onto variables  $\mathbf{y} \subseteq \mathbf{x}$ .

### 3 Integer Recursive Programs

We consider in the following that programs are collections of procedures that call each other, possibly according to recursive schemes. Formally, an *integer program* is an indexed tuple  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$ , where  $P_1, \dots, P_n$  of *procedures*. Each procedure is a tuple  $P_i = \langle \mathbf{x}_i, \vec{\mathbf{x}}_i^{\text{in}}, \vec{\mathbf{x}}_i^{\text{out}}, S_i, q_i^{\text{init}}, F_i, \Delta_i \rangle$ , where  $\mathbf{x}_i$  are the *local variables*<sup>3</sup> of  $P_i$  ( $\mathbf{x}_i \cap \mathbf{x}_j = \emptyset$  for all  $i \neq j$ ),  $\vec{\mathbf{x}}_i^{\text{in}}, \vec{\mathbf{x}}_i^{\text{out}} \subseteq \mathbf{x}_i$  are the ordered tuples of input and output variables,  $S_i$  are the *control states* of  $P_i$  ( $S_i \cap S_j = \emptyset$ , for all  $i \neq j$ ),  $q_i^{\text{init}} \in S_i$  is the *initial*, and  $F_i \subseteq S_i$  are the *final states* of  $P_i$ , and  $\Delta_i$  is a set of *transitions* of one of the following forms:

- $q \xrightarrow{\mathcal{R}(\mathbf{x}_i, \mathbf{x}_i')} q'$  is an *internal transition*, where  $q, q' \in S_i$ , and  $\mathcal{R}(\mathbf{x}_i, \mathbf{x}_i')$  is a Presburger arithmetic relation involving only the local variables of  $P_i$
- $q \xrightarrow{\vec{\mathbf{z}}' = P_j(\vec{\mathbf{u}})} q'$  is a *call*, where  $q, q' \in S_i$ ,  $P_j$  is the callee,  $\vec{\mathbf{u}}$  are linear terms over  $\mathbf{x}_i$ ,  $\vec{\mathbf{z}} \subseteq \mathbf{x}_j$  are variables, such that  $|\vec{\mathbf{u}}| = |\vec{\mathbf{x}}_j^{\text{in}}|$  and  $|\vec{\mathbf{z}}| = |\vec{\mathbf{x}}_j^{\text{out}}|$ .

The *call graph* of a program  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  is a directed graph with vertices  $P_1, \dots, P_n$  and edges  $(P_i, P_j)$  if and only if  $P_i$  has a call to  $P_j$ . A program is said to be *recursive* if its call graph has at least one cycle, and *non-recursive* if its call graph is a dag. Finally, let  $n\mathcal{F}(P_i)$  denotes the set  $S_i \setminus F_i$  of non-final state of  $P_i$ . Also  $n\mathcal{F}(\mathcal{P}) = \bigcup_{i=1}^n (S_i \setminus F_i)$ .

**Simplified syntax** To ease the description of programs defined in this paper, we use a simplified, human readable, imperative language such that each procedure of the program conforms to the following grammar:<sup>4</sup>

$P ::= \text{proc } P_i(id^*) \text{ begin var } id^* S \text{ end}$

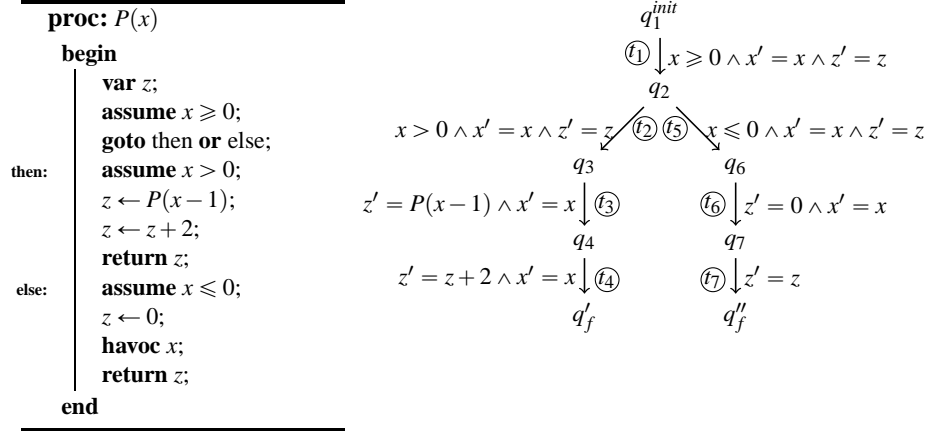
$S ::= S; S \mid \text{assume } f \mid id^n \leftarrow t^n \mid id \leftarrow P_i(t^*) \mid P_i(t^*) \mid \text{return } (id + \varepsilon) \mid \text{goto } \ell^+ \mid \text{havoc } id^+$

The local variables occurring in  $P$  are denoted by  $id$ , linear terms by  $t$ , Presburger formulae by  $f$ , and control labels by  $\ell$ . Each procedure consists in local declarations followed by a sequence of statements. Statements may carry a label. Program statements can be either assume statements<sup>5</sup>, (parallel) assignments, procedure calls (possibly with a return value), return to the caller (possibly with a value), non-deterministic jumps and

<sup>3</sup> Observe that there are no global variables in the definition of integer program. Those can be encoded as input and output variables to each procedure.

<sup>4</sup> Our simplified syntax does not seek to capture the generality of integer programs. Instead, our goal is to give a convenient notation for the programs given in this paper and only those.

<sup>5</sup> **assume**  $f$  is executable if and only if the current values of the variables satisfy  $f$ .



**Fig. 1.** Example of a simplified imperative program and its integer program thereof

havoc statements<sup>6</sup>. We consider the usual syntactic requirements (used variables must be declared, jumps are well defined, no jumps outside procedures, etc.). We do not define them, it suffices to know that all simplified programs in this paper comply with the requirements. A program using the simplified syntax can be easily translated into the formal syntax, as shown at Fig. 1.

*Example 1.* Figure 1 shows a program in our simplified imperative language and its corresponding integer program  $\mathcal{P}$ . Formally,  $\mathcal{P} = \langle P \rangle$  where  $P$  is defined as:  $\langle \{x, z\}, \langle x \rangle, \langle z \rangle, \{q_1^{init}, q_2, q_3, q_4, q_6, q_7, q'_f, q''_f\}, q_1^{init}, \{q'_f, q''_f\}, \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\} \rangle$ . Since  $P$  calls itself ( $t_3$ ), this program is recursive. ■

**Semantics** We are interested in computing the *summary relation* between the values of the input and output variables of a procedure. To this end, we give the semantics of a program  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  as a tuple of relations  $R_q$  describing, for each non-final control state  $q \in n\mathcal{F}(\mathcal{P}_i)$ , the effect of the program when started in  $q$  upon reaching a state in  $F_i$ . An *interprocedurally valid path* is represented by a tagged word over an alphabet  $\hat{\Theta}$ , which maps each internal transition  $t$  to a symbol  $\tau$ , and each call transition  $t$  to a pair of symbols  $\langle \tau, \tau \rangle \in \hat{\Theta}$ . In the sequel, we denote by  $Q$  the variable corresponding to the control state  $q$ , and by  $\tau \in \Theta$  the alphabet symbol corresponding to the transition  $t$  of  $\mathcal{P}$ . Formally, we associate to  $\mathcal{P}$  a visibly pushdown grammar, denoted in the rest of the paper by  $G_{\mathcal{P}} = (\mathcal{X}, \hat{\Theta}, \delta)$ , such that:

- $Q \in \mathcal{X}$  if and only if  $q \in n\mathcal{F}(\mathcal{P})$ ;
- $Q \rightarrow \tau Q' \in \delta$  if and only if  $t: q \xrightarrow{\mathcal{R}} q'$  and  $q' \in n\mathcal{F}(\mathcal{P})$ ;
- $Q \rightarrow \tau \in \delta$  if and only if  $t: q \xrightarrow{\mathcal{R}} q'$  and  $q' \notin n\mathcal{F}(\mathcal{P})$ ;
- $Q \rightarrow \langle \tau Q_j^{init} \tau \rangle Q' \in \delta$  if and only if  $t: q \xrightarrow{\overline{\mathbf{z}}' = P_j(\overline{\mathbf{u}})} q'$ .

<sup>6</sup> **havoc**  $x_1, x_2, \dots, x_n$  assigns non deterministically chosen integers to  $x_1, x_2, \dots, x_n$ .

It is easily seen that interprocedurally valid paths in  $\mathcal{P}$  and tagged words in  $G_{\mathcal{P}}$  are in one-to-one correspondence. In fact, each interprocedurally valid path of  $\mathcal{P}$  between state  $q \in n\mathcal{F}(P_i)$  and a state of  $F_i$ , where  $1 \leq i \leq n$ , corresponds exactly to one tagged word of  $L_Q(G_{\mathcal{P}})$ .

*Example 2.* (continued from Ex. 1) The visibly pushdown grammar  $G_{\mathcal{P}}$  corresponding to  $\mathcal{P}$  consists of the following variables and labelled productions:

$$\begin{array}{ll} p_1^b \stackrel{\text{def}}{=} Q_1^{\text{init}} \rightarrow \tau_1 Q_2 & p_3^c \stackrel{\text{def}}{=} Q_3 \rightarrow \langle \tau_3 Q_1^{\text{init}} \tau_3 \rangle Q_4 \\ p_2^b \stackrel{\text{def}}{=} Q_2 \rightarrow \tau_2 Q_3 & p_4^a \stackrel{\text{def}}{=} Q_4 \rightarrow \tau_4 \\ p_5^b \stackrel{\text{def}}{=} Q_2 \rightarrow \tau_5 Q_6 & p_6^b \stackrel{\text{def}}{=} Q_6 \rightarrow \tau_6 Q_7 \\ & p_7^a \stackrel{\text{def}}{=} Q_7 \rightarrow \tau_7 \end{array}$$

$L_{Q_1^{\text{init}}}(G_{\mathcal{P}})$  includes the word  $w = \tau_1 \tau_2 \langle \tau_3 \tau_1 \tau_5 \tau_6 \tau_7 \tau_3 \rangle \tau_4$ , defining the nested word  $w_{\mathcal{NW}}(w) = (\tau_1 \tau_2 \tau_3 \tau_1 \tau_5 \tau_6 \tau_7 \tau_3 \tau_4, \{3 \rightsquigarrow 8\})$ . The word  $w$  corresponds to an interprocedurally valid path where  $P$  calls itself once. Let  $\gamma_1 = p_1^b p_2^b p_3^c p_4^a p_1^b p_5^b p_6^b p_7^a$  and  $\gamma_2 = p_1^b p_2^b p_3^c p_1^b p_5^b p_6^b p_7^a p_4^a$  be two control words we have  $Q_1^{\text{init}} \xRightarrow{\gamma_1} w$  and  $Q_1^{\text{init}} \xRightarrow{\gamma_2} w$ . ■

The semantics of a program is the union of the semantics of the nested words corresponding to its executions, each of the latter being a relation over input and output variables. To define the semantics of a nested word, we first associate to each  $\tau \in \hat{\Theta}$  an integer relation  $\rho_{\tau}$ , defined as follows:

- for an internal transition  $t : q \xrightarrow{\mathcal{R}} q' \in \Delta_i$ , let  $\rho_{\tau} \equiv \mathcal{R}(\mathbf{x}_i, \mathbf{x}'_i) \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$
- for a call transition  $t : q \xrightarrow{\overline{\mathbf{z}}' = P_j(\overline{\mathbf{u}})} q' \in \Delta_i$ , we define a *call relation*  $\rho_{\langle \tau \rangle} \equiv (\overline{\mathbf{x}}_j^{\text{in}'} = \overline{\mathbf{u}}) \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_j}$ , a *return relation*  $\rho_{\tau \rangle} \equiv (\overline{\mathbf{z}}' = \overline{\mathbf{x}}_j^{\text{out}}) \subseteq \mathbb{Z}^{\mathbf{x}_j} \times \mathbb{Z}^{\mathbf{x}_i}$  and a *frame relation*  $\phi_{\tau} \equiv \bigwedge_{x \in \mathbf{x}_i} \overline{\mathbf{z}} x' = x \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$

We define the semantics of the program  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  in a top-down manner. Assuming a fixed ordering of the non-final states in the program, i.e.  $n\mathcal{F}(\mathcal{P}) = \langle q_1, \dots, q_m \rangle$ , the semantics of the program  $\mathcal{P}$ , denoted  $\llbracket \mathcal{P} \rrbracket$ , is the tuple of relations  $\langle \llbracket q_1 \rrbracket, \dots, \llbracket q_m \rrbracket \rangle$ . For each non-final control state  $q \in n\mathcal{F}(P_i)$  where  $1 \leq i \leq n$ , we denote by  $\llbracket q \rrbracket \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$  the relation (over the local variables of procedure  $P_i$ ) defined as  $\llbracket q \rrbracket = \bigcup_{\alpha \in L_Q(G_{\mathcal{P}})} \llbracket \alpha \rrbracket$ .

It remains to define  $\llbracket \alpha \rrbracket$ , the semantics of the tagged word  $\alpha$ . Because it is more convenient, we define the semantics of its corresponding nested word  $w_{\mathcal{NW}}(\alpha) = (\tau_1 \dots \tau_{\ell}, \rightsquigarrow)$  over alphabet  $\hat{\Theta}$ . For a nesting relation  $\rightsquigarrow \subseteq \{1, \dots, \ell\} \times \{1, \dots, \ell\}$ , we denote by  $\rightsquigarrow_{i,j}$  the relation  $\{(s - (i - 1), t - (i - 1)) \mid (s, t) \in \rightsquigarrow \cap \{i, \dots, j\} \times \{i, \dots, j\}\}$ , for some  $i, j \in \{1, \dots, \ell\}$ ,  $i < j$ . Finally, we define  $\llbracket (\tau_1 \dots \tau_{\ell}, \rightsquigarrow) \rrbracket \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$  (recall that  $\alpha \in L_Q(G_{\mathcal{P}})$  and  $q$  is a state of  $P_i$ ) as follows:

$$\llbracket (\tau_1 \dots \tau_{\ell}, \rightsquigarrow) \rrbracket = \begin{cases} \rho_{\tau_1} & \text{if } \ell = 1 \\ \rho_{\tau_1} \circ \llbracket (\tau_2 \dots \tau_{\ell}, \rightsquigarrow_{2,\ell}) \rrbracket & \text{if } \ell > 1 \text{ and } 1 \not\rightsquigarrow j \text{ for all } 1 \leq j \leq \ell \\ R_{\tau} \circ \llbracket (\tau_{j+1} \dots \tau_{\ell}, \rightsquigarrow_{j+1,\ell}) \rrbracket & \text{if } \ell > 1 \text{ and } 1 \rightsquigarrow j \text{ for some } 1 \leq j \leq \ell \end{cases}$$

where, in the last case, which corresponds to call transition  $t : q \xrightarrow{\overline{\mathbf{z}}' = P_d(\overline{\mathbf{u}})} q' \in \Delta_i$ , we have  $\tau_1 = \tau_j = \tau$  and  $R_{\tau} = (\rho_{\langle \tau \rangle} \circ \llbracket \tau_2 \dots \tau_{j-1}, \rightsquigarrow_{2,j-1} \rrbracket) \circ \rho_{\tau \rangle} \cap \phi_{\tau}$ .

*Example 3.* (continued from Ex. 2) Given the nested word  $\theta = (\tau_1\tau_2\tau_3\tau_1\tau_5\tau_6\tau_7\tau_3\tau_4, \{3 \rightsquigarrow 8\})$  its semantics,  $\llbracket \theta \rrbracket$ , is a relation between valuations of  $\{x, z\}$ , given by:

$$\rho_{\tau_1} \circ \rho_{\tau_2} \circ ((\rho_{\tau_3} \circ \rho_{\tau_1} \circ \rho_{\tau_5} \circ \rho_{\tau_6} \circ \rho_{\tau_7} \circ \rho_{\tau_3}) \cap \phi_{\tau_3}) \circ \rho_{\tau_4}$$

One can verify that  $\llbracket \theta \rrbracket \equiv x = 1 \wedge z' = 2$ , i.e. the result of calling  $P$  with an input valuation  $x = 1$  is the output valuation  $z = 2$ . ■

Finally, we introduce a few useful notations. By  $\llbracket \mathcal{P} \rrbracket_q$  we denote the component of  $\llbracket \mathcal{P} \rrbracket$  corresponding to  $q \in n\mathcal{F}(\mathcal{P})$ . Slightly abusing notations we define  $L_{P_i}(G_{\mathcal{P}})$  as  $L_{Q_i^{init}}(G_{\mathcal{P}})$  and  $\llbracket \mathcal{P} \rrbracket_{P_i}$  as  $\llbracket \mathcal{P} \rrbracket_{q_i^{init}}$ . Finally, define  $\llbracket \mathcal{P} \rrbracket_{P_i}^{i/o} = \{\langle I \downarrow_{x_i^{in}}, O \downarrow_{x_i^{out}} \rangle \mid \langle I, O \rangle \in \llbracket \mathcal{P} \rrbracket_{P_i}\}$ .

## 4 Underapproximating the Program Semantics

In the section we define a family of underapproximations of  $\llbracket \mathcal{P} \rrbracket$  called bounded-index underapproximations. Then we show that each  $k$ -index underapproximation of the semantics of a (possibly recursive) program  $\mathcal{P}$  coincides with the semantics of a non-recursive program computable from  $\mathcal{P}$  and  $k$ . The central notion of bounded-index derivation is introduced in the following followed by basic properties about them.

**Definition 1.** Given a grammar  $G$  with relation  $\Longrightarrow$  between strings, for every  $k \geq 1$  we define the subrelation  $\Longrightarrow^{(k)}$  of  $\Longrightarrow$  as follows:  $u \Longrightarrow^{(k)} v$  iff  $u \Longrightarrow v$  and both  $u$  and  $v$  contain at most  $k$  occurrences of variables. We denote by  $\Longrightarrow^{(k)*}$  the reflexive transitive closure of  $\Longrightarrow^{(k)}$ . Hence given  $X$  and  $k$  define  $L_X^{(k)}(G) = \{w \in \Sigma^* \mid X \Longrightarrow^{(k)*} w\}$  and we call the derivation of  $w \in \Sigma^*$  from  $X$  a  $k$ -index derivation. A grammar  $G$  is said to have index  $k$  whenever  $L_X(G) = L_X^{(k)}(G)$  for each  $X \in \mathcal{X}$ .<sup>7</sup>

**Lemma 1.** For every grammar the following properties hold: (1)  $\Longrightarrow^{(k)} \subseteq \Longrightarrow^{(k+1)}$  for all  $k \geq 1$ ; (2)  $\Longrightarrow = \bigcup_{k=1}^{\infty} \Longrightarrow^{(k)}$ ; (3)  $BC \Longrightarrow^{(k)*} w \in \Sigma^*$  iff there exist  $w_1, w_2$  such that  $w = w_1 w_2$  and either (i)  $B \Longrightarrow^{(k-1)*} w_1$ ,  $C \Longrightarrow^{(k)*} w_2$ , or (ii)  $C \Longrightarrow^{(k-1)*} w_2$  and  $B \Longrightarrow^{(k)*} w_1$ .

The main intuition behind our method is to filter out interprocedurally valid paths which can not be produced by  $k$ -index derivations. Our analysis is then carried out on the remaining paths produced by  $k$ -index derivations only. We argue that this underapproximation technique is more general than bounding the stack space of the program which corresponds to filter out derivations which are either non *leftmost*<sup>8</sup> or not  $k$ -index.

*Example 4.* (continued from Ex. 2)  $P$  is a (non-tail) recursive procedure and  $G_{\mathcal{P}}$  models its control flow. Inspecting  $G_{\mathcal{P}}$  reveals that  $L_{Q_1^{init}}(G_{\mathcal{P}}) = \{(\tau_1\tau_2\langle\tau_3\rangle^n\tau_1\tau_5\tau_6\tau_7\langle\tau_3\rangle\tau_4)^n \mid n \geq 0\}$ . For each value of  $n$  we give a 2-index derivation capturing the word: repeat

<sup>7</sup> Gruska [17] proved that deciding whether  $L_X(G) = L_X^{(k)}(G)$  for some  $k \geq 1$  is undecidable.

<sup>8</sup> A leftmost derivation is a derivation where, at each step, the production that is applied rewrites the leftmost nonterminal.

$n$  times the steps  $Q_1^{init} \xrightarrow{p_1^b p_2^b p_3^c} \tau_1 \tau_2 \langle \tau_3 Q_1^{init} \tau_3 \rangle Q_4 \xrightarrow{p_4^a} \tau_1 \tau_2 \langle \tau_3 Q_1^{init} \tau_3 \rangle \tau_4$  followed by the steps  $Q_1^{init} \xrightarrow{p_1^b p_2^b p_3^b p_4^a} \tau_1 \tau_5 \tau_6 \tau_7$ . Therefore the 2-index approximation of  $G_{\mathcal{P}}$  shows that  $L_{Q_1^{init}}(G_{\mathcal{P}}) = L_{Q_1^{init}}^{(2)}(G_{\mathcal{P}})$ . However bounding the number of times  $P$  calls itself up to 2 results in 3 interprocedurally valid paths (for  $n = 0, 1, 2$ ).

Given  $k \geq 1$ , we define the  $k$ -index semantics of  $\mathcal{P}$  as  $\llbracket \mathcal{P} \rrbracket^{(k)} = \langle \llbracket q_1 \rrbracket^{(k)}, \dots, \llbracket q_m \rrbracket^{(k)} \rangle$ , where the  $k$ -index semantics of a non-final control state  $q$  of a procedure  $P_i$  is the relation  $\llbracket q \rrbracket^{(k)} \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$ , defined as  $\llbracket q \rrbracket = \bigcup_{\alpha \in L_Q^{(k)}(G_{\mathcal{P}})} \llbracket \alpha \rrbracket$ .

#### 4.1 Computing Bounded-index Underapproximations

In what follows, we define a source-to-source transformation that takes in input a recursive program  $\mathcal{P}$ , an integer  $k \geq 1$  and returns a *non-recursive* program  $\mathcal{H}^k$  which has the same semantics as  $\llbracket \mathcal{P} \rrbracket^{(k)}$  (modulo projection on some variables). Therefore every off-the-shelf tool, that computes the summary semantics for a non-recursive program, can be used to compute the  $k$ -index semantics of  $\mathcal{P}$ , for any given  $k \geq 1$ .

Let  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  be a program, and  $\mathbf{x} = \bigcup_{i=1}^n \mathbf{x}_i$  be the set of all variables in  $\mathcal{P}$ . As we did previously, we assume a fixed ordering  $\langle q_1, \dots, q_m \rangle$  on the set  $n\mathcal{F}(\mathcal{P})$ . Let  $G_{\mathcal{P}} = (\mathcal{X}, \hat{\Theta}, \delta)$  be the visibly pushdown grammar associated with  $\mathcal{P}$ , such that each non-final state  $q$  of  $\mathcal{P}$  is associated a nonterminal  $Q \in \mathcal{X}$ . Then we define a *non-recursive* program  $\mathcal{H}^K$  that captures the  $K$ -index semantics of  $\mathcal{P}$  (Algorithm 1), for  $K \geq 1$ . Formally, we define  $\mathcal{H}^K = \times_{k=0}^K \langle query_{Q_1}^k, \dots, query_{Q_m}^k \rangle$ , where:

- for each  $k = 0, \dots, K$  and each control state  $q \in n\mathcal{F}(\mathcal{P})$ , we have a procedure  $query_Q^k$ ;
- in particular,  $query_{Q_1}^0, \dots, query_{Q_m}^0$  consists of one **assume false** statement;
- each procedure  $query_Q^k$  has five sets of local variables, all of the same cardinality as  $\mathbf{x}$ : two sets, named  $\mathbf{x}_I$  and  $\mathbf{x}_O$ , are used as input variables, whereas the other three sets, named  $\mathbf{x}_J, \mathbf{x}_K$  and  $\mathbf{x}_L$  are used locally by  $query_Q^k$ . Besides,  $query_Q^k$  has a local variable called  $PC$ . There are no output variables.

Observe that each procedure  $query_Q^k$  calls only procedures  $query_{Q'}^{k-1}$  for some  $Q'$ , hence the program  $\mathcal{H}^K$  is non-recursive, and therefore amenable to summarization techniques that cannot handle recursion. Also the hierarchical structure of  $\mathcal{H}^K$  enables modular summarization by computing the summaries ordered by increasing values of  $k = 0, 1, \dots, K$ . The summaries of  $\mathcal{H}^{K-1}$  are reused to compute  $\mathcal{H}^K$ . Finally, it is routine to check that the size of  $\mathcal{H}^K$  (viz. the number of statements) is in  $O(K \cdot \#Prod)$  where  $\#Prod$  is the number of productions of  $G_{\mathcal{P}}$ . Consequently the time needed to generate  $\mathcal{H}^K$  is linear in the product  $K \cdot \#Prod$ .

Given that  $query_Q^k$  has two copies of  $\mathbf{x}$  as input variables, and no output variables, the input output semantics  $\llbracket \mathcal{H}^k \rrbracket_{query_Q^k}^{i/o} \subseteq \mathbb{Z}^{\mathbf{x} \times \mathbf{x}}$  is a set of tuples, rather than a (binary) relation. Given two valuations  $I, O \in \mathbb{Z}^{\mathbf{x}}$ , we denote by  $I \cdot O \in \mathbb{Z}^{\mathbf{x} \times \mathbf{x}}$  their concatenation.

Thm. 1 relates the semantics of  $\mathcal{H}^K$  and the  $K$ -index semantics of  $\mathcal{P}$ . Given  $k$ ,  $1 \leq k \leq K$  and a control state  $q$  of  $\mathcal{P}$ , we show equality between  $\llbracket \mathcal{H}^K \rrbracket_{query_Q^k}^{i/o}$  and  $\llbracket \mathcal{P} \rrbracket_q^{(k)}$  over common variables. Before starting, we fix an arbitrary value for  $K$  and require that each  $k$  is such that  $1 \leq k \leq K$ . Hence, we drop  $K$  in  $\mathcal{H}^K$  and write  $\mathcal{H}$ .



---

**Algorithm 1:**  $\text{proc } \text{query}_Q^k(\mathbf{x}_I, \mathbf{x}_O)$  for  $k \geq 1$

---

```

begin
  var  $PC, \mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$ ;
   $PC \leftarrow Q$ ;
start: goto  $p_1^a$  or  $\dots$  or  $p_{n_a}^a$  or  $p_1^b$  or  $\dots$  or  $p_{n_b}^b$  or  $p_1^c$  or  $\dots$  or  $p_{n_c}^c$ ;
 $p_1^a$ : assume ( $PC = \text{head}(p_1^a)$ ); assume  $\rho_{\text{tail}(p_1^a)}(\mathbf{x}_I, \mathbf{x}_O)$ ; return;
       $\vdots$ 
 $p_{n_a}^a$ : assume ( $PC = \text{head}(p_{n_a}^a)$ ); assume  $\rho_{\text{tail}(p_{n_a}^a)}(\mathbf{x}_I, \mathbf{x}_O)$ ; return;
 $p_1^b$ : assume ( $PC = \text{head}(p_1^b)$ ); [ paste code for case  $\text{tail}(p_1^b) \in \Theta \times \mathcal{X}$  ];
       $\vdots$ 
 $p_{n_b}^b$ : assume ( $PC = \text{head}(p_{n_b}^b)$ ); [ paste code for case  $\text{tail}(p_{n_b}^b) \in \Theta \times \mathcal{X}$  ];
 $p_1^c$ : assume ( $PC = \text{head}(p_1^c)$ ); [ paste code for case  $\text{tail}(p_1^c) \in \langle \Theta \times \mathcal{X} \times \Theta \rangle \times \mathcal{X}$  ];
       $\vdots$ 
 $p_{n_c}^c$ : assume ( $PC = \text{head}(p_{n_c}^c)$ ); [ paste code for case  $\text{tail}(p_{n_c}^c) \in \langle \Theta \times \mathcal{X} \times \Theta \rangle \times \mathcal{X}$  ];
end

```

---

<p><b>case:</b> <math>\text{tail}(p_i^b) = \tau Q' \in \Theta \times \mathcal{X}</math></p> <p><b>havoc</b> (<math>\mathbf{x}_J</math>);</p> <p><b>assume</b> <math>\rho_\tau(\mathbf{x}_I, \mathbf{x}_J)</math>;</p> <p><math>\mathbf{x}_J \leftarrow \mathbf{x}_J</math>;</p> <p><math>PC \leftarrow Q'</math>; // <math>\text{query}_{Q'}^k(\mathbf{x}_I, \mathbf{x}_O)</math></p> <p><b>goto start</b>; // <b>return</b></p>	<p><b>case:</b> <math>\text{tail}(p_i^c) = \langle \tau Q_j^{\text{init}} \tau \rangle Q' \in \langle \Theta \times \mathcal{X} \times \Theta \rangle \times \mathcal{X}</math></p> <p><b>havoc</b> (<math>\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L</math>);</p> <p><b>assume</b> <math>\rho_{\langle \tau \rangle}(\mathbf{x}_I, \mathbf{x}_J)</math>; /* call relation */</p> <p><b>assume</b> <math>\rho_\tau(\mathbf{x}_K, \mathbf{x}_L)</math>; /* return relation */</p> <p><b>assume</b> <math>\phi_\tau(\mathbf{x}_I, \mathbf{x}_L)</math>; /* frame relation */</p> <p><b>goto ord or rod</b>;</p>
<p>In Alg. 1, <math>p_i^\alpha</math> where <math>\alpha \in \{a, b, c\}</math> <b>ord:</b></p> <p>refers to a production of the visibly pushdown grammar <math>G_{\mathcal{P}}</math>.</p> <p>The same symbol in boldface refers to the labelled statements <b>rod:</b></p> <p>in Alg. 1. The superscript <math>\alpha \in \{a, b, c\}</math> differentiate the productions whether they are the form <math>Q \rightarrow \tau</math>, <math>Q \rightarrow \tau Q'</math> or <math>Q \rightarrow \langle \tau Q_j^{\text{init}} \tau \rangle Q'</math>, respectively.</p>	<p><math>\text{query}_{Q_j^{\text{init}}}^{k-1}(\mathbf{x}_J, \mathbf{x}_K)</math>; /* in order exec. */</p> <p><math>\mathbf{x}_I \leftarrow \mathbf{x}_L</math>;</p> <p><math>PC \leftarrow Q'</math>; // <math>\text{query}_{Q'}^k(\mathbf{x}_I, \mathbf{x}_O)</math></p> <p><b>goto start</b>; // <b>return</b></p> <p><math>\text{query}_{Q'}^{k-1}(\mathbf{x}_L, \mathbf{x}_O)</math>; /* out of order exec. */</p> <p><math>\mathbf{x}_I \leftarrow \mathbf{x}_J</math>;</p> <p><math>\mathbf{x}_O \leftarrow \mathbf{x}_K</math>;</p> <p><math>PC \leftarrow Q_j^{\text{init}}</math>; // <math>\text{query}_{Q_j^{\text{init}}}^k(\mathbf{x}_I, \mathbf{x}_O)</math></p> <p><b>goto start</b>; // <b>return</b></p>

---

One way to prove Thm. 1 consists in first unfolding the definitions of the semantics as follows:  $\llbracket \mathcal{H} \rrbracket_{\text{query}_Q^k} = \bigcup_{\alpha \in L_{\text{query}_Q^k}(G_{\mathcal{H}})} \llbracket \alpha \rrbracket$ ,  $\llbracket \mathcal{P} \rrbracket_q^{(k)} = \bigcup_{\beta \in L_Q^{(k)}(G_{\mathcal{P}})} \llbracket \beta \rrbracket$  then establish a relationship between the  $\alpha$ 's and the  $\beta$ 's that implies the equivalence of their semantics over common variables. Instead, we follow an equivalent, but more intuitive, approach in which the semantics of  $\mathcal{H}$  is obtained by interpreting directly its code. After all, the interprocedurally valid paths in procedure  $\text{query}_Q^k$  are in one-to-one correspondence with the words of  $L_{\text{query}_Q^k}(G_{\mathcal{H}})$ .

An inspection of the code of  $\mathcal{H}$  reveals that  $\mathcal{H}$  simulates  $k$ -index depth first derivations of  $G_{\mathcal{P}}$  and interprets the statements of  $\mathcal{P}$  on its local variables while applying

derivation steps. By considering non necessarily leftmost derivations  $\mathcal{H}$  interprets the statements of  $\mathcal{P}$  in an order which differs from the expected one.

*Example 5.* Let us consider an execution of *query* for the call  $query_{Q_1}^2((1\ 0), (1\ 2))$

following  $Q_1^{init} \xrightarrow{p_1^b p_2^b p_3^c} \tau_1 \tau_2 \langle \tau_3 Q_1^{init} \tau_3 \rangle Q_4 \xrightarrow{p_4^a} \tau_1 \tau_2 \langle \tau_3 Q_1^{init} \tau_3 \rangle \tau_4 \xrightarrow{p_1^b p_3^c p_6^b p_7^a} \tau_1 \tau_2 \langle \tau_3 \tau_1 \tau_5 \tau_6 \tau_7 \tau_3 \rangle \tau_4$ . In the table below, the first row (labelled  $k/PC$ ) gives the caller ( $1 = query_{Q_4}^1$ ,  $2 = query_{Q_1}^2$ ) and the value of PC when control hits the labelled statement given at the second row (labelled  $ip$ ). The third row (labelled  $\mathbf{x}_I/\mathbf{x}_O$ ) represents the content of the two arrays.  $\mathbf{x}_I/\mathbf{x}_O = (a\ b)(c\ d)$  says that, in  $\mathbf{x}_I$ ,  $x$  has value  $a$  and  $z$  has value  $b$ ; in  $\mathbf{x}_O$ ,  $x$  has value  $c$  and  $z$  has value  $d$ .

$k/PC$	$2/Q_1^{init}$	$2/Q_1^{init}$	$2/Q_2$	$2/Q_2$	$2/Q_3$	$2/Q_3$	$2/Q_3$
$ip$	<b>start</b>	<b>p<sub>1</sub><sup>b</sup></b>	<b>start</b>	<b>p<sub>2</sub><sup>b</sup></b>	<b>start</b>	<b>p<sub>3</sub><sup>c</sup></b>	<b>rod</b>
$\mathbf{x}_I/\mathbf{x}_O$	(1 0)(1 2)	(1 0)(1 2)	(1 0)(1 2)	(1 0)(1 2)	(1 0)(1 2)	(1 0)(1 2)	(1 0)(1 2)
$k/PC$	$1/Q_4$	$1/Q_4$	$2/Q_1^{init}$	$2/Q_1^{init}$	$2/Q_2$	$2/Q_2$	$2/Q_6$
$ip$	<b>start</b>	<b>p<sub>4</sub><sup>a</sup></b>	<b>start</b>	<b>p<sub>1</sub><sup>b</sup></b>	<b>start</b>	<b>p<sub>5</sub><sup>b</sup></b>	<b>start</b>
$\mathbf{x}_I/\mathbf{x}_O$	(1 0)(1 2)	(1 0)(1 2)	(0 0)(42 0)	(0 0)(42 0)	(0 0)(42 0)	(0 0)(42 0)	(0 0)(42 0)
$k/PC$	$2/Q_6$	$2/Q_7$	$2/Q_7$				
$ip$	<b>p<sub>6</sub><sup>b</sup></b>	<b>start</b>	<b>p<sub>7</sub><sup>a</sup></b>				
$\mathbf{x}_I/\mathbf{x}_O$	(0 0)(42 0)	(0 0)(42 0)	(0 0)(42 0)				

The execution of  $query_{Q_1}^2$  starts on row 1, column 1 and proceeds until the call to  $query_{Q_4}^1$  at row 2, column 1 (the out of order case). The latter ends at row 2, column 2, where the execution of  $query_{Q_1}^2$  resumes. Since the execution is out of order, and the previous **havoc**( $\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$ ) results into  $\mathbf{x}_J = (0\ 0)$ ,  $\mathbf{x}_K = (42\ 0)$  and  $\mathbf{x}_L = (1\ 0)$  (this choice complies with the call relation), the values of  $\mathbf{x}_I/\mathbf{x}_O$  are updated to  $(0\ 0)/(42\ 0)$ . The choice for equal values (0) of  $z$  in both  $\mathbf{x}_I$  and  $\mathbf{x}_O$  is checked in row 3, column 3. ■

**Theorem 1.** Let  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  be a program and let  $q \in n\mathcal{F}(P_i)$  be a non-final control state of some  $P_i = \langle \mathbf{x}_i, \vec{\mathbf{x}}_i^{in}, \vec{\mathbf{x}}_i^{out}, S_i, q_i^{init}, F_i, \Delta_i \rangle$ . Then, for any  $k \geq 1$ , we have:

$$\llbracket \mathcal{H} \rrbracket_{query_Q^k}^{i/o} = \{ I \cdot O \in \mathbb{Z}^{\mathbf{x} \times \mathbf{x}} \mid \langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)} \} .$$

Consequently, we also have:

$$\llbracket \mathcal{P} \rrbracket_q^{(k)} = \{ \langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \mid I \cdot O \in \llbracket \mathcal{H} \rrbracket_{query_Q^k}^{i/o} \} .$$

The proof of Thm. 1 is based on the following lemma.

**Lemma 2.** Let  $k \geq 1$ ,  $q$  be a non-final control state of  $P_i$  and  $I, O \in \mathbb{Z}^{\mathbf{x}}$ . If the call to  $query_Q^k(I, O)$  returns then  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$ . Conversely, if  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$  then there exists  $I', O' \in \mathbb{Z}^{\mathbf{x}}$  such that  $I' \downarrow_{\mathbf{x}_i} = I \downarrow_{\mathbf{x}_i}$ ,  $O' \downarrow_{\mathbf{x}_i} = O \downarrow_{\mathbf{x}_i}$  and  $query_Q^k(I', O')$  returns.

*Proof:* First we consider a tail-recursive version of Algorithm 1 which is obtained by replacing every two statements of the form  $PC \leftarrow X$ ; **goto start**; by  $query_X^k(\mathbf{x}_I, \mathbf{x}_O)$ ; **return**; (as it appears in the comments of Alg. 1). The equivalence between Algorithm 1 and its tail-recursive variant is an easy exercise.

“ $\Leftarrow$ ” Let  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$ . By definition of  $k$ -index semantics, there exists  $\alpha \in L_Q^{(k)}(G_{\mathcal{P}})$  such that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \alpha \rrbracket$ . Let  $p_1$  be the first production used in the derivation of  $\alpha$  and let  $\ell \geq 1$  be the length (in number of productions used) of the derivation. Our proof proceeds by induction on  $\ell$ . If  $\ell = 1$  then we find that  $p_1$  must be of the form  $Q \rightarrow \tau$  and that  $\alpha = \tau$ . Therefore we have  $\llbracket \alpha \rrbracket = \llbracket \tau \rrbracket = \rho_\tau$  and moreover  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \rho_\tau$ . Since  $k \geq 1$ , we let  $I' = I$  and  $O' = O$  we find that  $query_Q^{(k)}(I', O')$  returns by choosing to jump to the label corresponding to  $p_1$ , then executing the **assume** statement and finally the **return** statement. When  $\ell > 1$ , the proof divides in two parts.

1. If  $p_1$  is of the form  $Q \rightarrow \tau Q'$  then we find that  $\alpha = \tau \beta$ . Moreover,  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \alpha \rrbracket = \rho_\tau \circ \llbracket \beta \rrbracket$  by definition of the semantics. This implies that there exists  $J \in \mathbb{Z}^{\mathbf{x}}$  such that  $\langle I \downarrow_{\mathbf{x}_i}, J \downarrow_{\mathbf{x}_i} \rangle \in \rho_\tau$  and  $\langle J \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \beta \rrbracket$ . Hence, we conclude from  $\beta \in L_{Q'}^{(k)}(G_{\mathcal{P}})$ , that  $\langle J \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_{Q'}^{(k)}$ . Applying the induction hypothesis on this last fact, we find that the call  $query_{Q'}^{(k)}(J, O)$  returns. Finally consider the call  $query_Q^k(I, O)$  where at label **start** the jump goes to label corresponding to  $p_1$ . At this point in the execution **havoc**( $\mathbf{x}_J$ ) returns  $J$ . Next **assume**  $\rho_\tau(I, J)$  succeeds. Finally we find that the call to  $query_Q^k(I, O)$  returns because so does the call  $query_{Q'}^k(J, O)$  which is followed by **return**.

2. If  $p_1$  is of the form  $Q \rightarrow \langle \tau Q_j^{init} \tau \rangle Q'$  then we find that  $\alpha = \langle \tau \beta' \tau \rangle \beta$  for some  $\beta', \beta$ . Lemma 1 (prop. 3) shows that either  $\beta' \in L_{Q_j^{init}}^{(k-1)}(G_{\mathcal{P}})$  and  $\beta \in L_{Q'}^{(k)}(G_{\mathcal{P}})$  or  $\beta' \in L_{Q_j^{init}}^{(k)}(G_{\mathcal{P}})$  and  $\beta \in L_{Q'}^{(k-1)}(G_{\mathcal{P}})$ . We will assume the former case, the latter being treated similarly. Moreover,  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \alpha \rrbracket = R_\tau \circ \llbracket \beta \rrbracket$ . The leftmost relation can be rewritten,  $\left( (\rho_{\langle \tau \rangle} \circ \llbracket \beta' \rrbracket \circ \rho_{\tau \rangle}) \cap \phi_\tau \right) \circ \llbracket \beta \rrbracket$  which by definition of  $\beta, \beta'$  and the semantics is included in  $\left( (\rho_{\langle \tau \rangle} \circ \llbracket \mathcal{P} \rrbracket_{Q_j^{init}}^{(k-1)} \circ \rho_{\tau \rangle}) \cap \phi_\tau \right) \circ \llbracket \mathcal{P} \rrbracket_{Q'}^{(k)}$ . We conclude from the previous relation that there exists  $J, K, L \in \mathbb{Z}^{\mathbf{x}}$  such that  $\langle I \downarrow_{\mathbf{x}_i}, J \downarrow_{\mathbf{x}_i} \rangle \in \rho_{\langle \tau \rangle}$ ,  $\langle J \downarrow_{\mathbf{x}_i}, K \downarrow_{\mathbf{x}_j} \rangle \in \llbracket \mathcal{P} \rrbracket_{Q_j^{init}}^{(k-1)}$ ,  $\langle K \downarrow_{\mathbf{x}_j}, L \downarrow_{\mathbf{x}_j} \rangle \in \rho_{\tau \rangle}$ , and  $\langle L \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_{Q'}^{(k)}$ . Applying the induction hypothesis we obtain that the calls  $query_{Q_j^{init}}^{k-1}(J, K)$  and  $query_{Q'}^k(L, O)$  return. Given those facts, it is routine to check that  $query_Q^k(I', O')$  returns by choosing to jump to label corresponding to  $p_1$ , then having **havoc**( $\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$ ) return  $(J, K, L)$  and we are done.

The proof for the only if direction is in appendix A.2.  $\square$

As a last point, we prove that the bounded-index sequence  $\{\llbracket \mathcal{P} \rrbracket^{(k)}\}_{k=1}^\infty$  satisfies several conditions that advocate for its use in program analysis, as an underapproximation sequence. The subset order and set union is extended to tuples of relations, point-wise.

$$\llbracket \mathcal{P} \rrbracket^{(k)} \subseteq \llbracket \mathcal{P} \rrbracket^{(k+1)} \quad \text{for all } k \geq 1 \quad (A1)$$

$$\llbracket \mathcal{P} \rrbracket = \bigcup_{k=1}^\infty \llbracket \mathcal{P} \rrbracket^{(k)} \quad (A2)$$

Condition (A1) requires that the sequence is monotonically increasing, the limit of this increasing sequence being the actual semantics of the program (A2). These conditions follow however immediately from the two first points of Lemma 1. To decide whether the limit  $\llbracket \mathcal{P} \rrbracket$  has been reached by some iterate  $\llbracket \mathcal{P} \rrbracket^{(k)}$ , it is enough to check that the

tuple of relations in  $\llbracket \mathcal{P} \rrbracket^{(k)}$  is inductive with respect to the statements of  $\mathcal{P}$ . This can be implemented as an SMT query.

## 5 Completeness of Underapproximations for Bounded Programs

In this section we define a class of recursive programs such that the precise summary semantics of each program in that class is effectively computable. We show for each program  $\mathcal{P}$  in the class that (a)  $\llbracket \mathcal{P} \rrbracket = \llbracket \mathcal{P} \rrbracket^{(k)}$  for some value of  $k \geq 1$ , and moreover (b) the semantics of  $\mathcal{H}^k$  is effectively computable (and so is that of  $\llbracket \mathcal{P} \rrbracket^{(k)}$  by Thm. 1).

**Periodic Relations** Given an integer relation  $R \subseteq \mathbb{Z}^n \times \mathbb{Z}^n$ , we define  $R^0$  as the identity relation on  $\mathbb{Z}^n$ , and  $R^{i+1} = R^i \circ R$ , for all  $i \geq 0$ . The *closed form* of  $R$  is a formula defining a relation  $\hat{R} \subseteq \mathbb{N} \times \mathbb{Z}^n \times \mathbb{Z}^n$  such that, for each  $n \geq 0$  we have  $\hat{R}(n) = R^n$ . In general, the closed form of a relation is not definable within decidable subsets of integer arithmetic, such as Presburger arithmetic. In this section we consider two classes of relations, called *periodic*, for which this is possible, namely octagonal relations, and finite monoid affine relations. The formal definitions are deferred to appendix A.3.

**Bounded languages** We define a *bounded-expression*  $\mathbf{b}$  to be a regular expression of the form  $\mathbf{b} = w_1^* \dots w_k^*$ , where  $k \geq 1$  and each  $w_i$  is a non-empty word. A language (not necessarily context-free)  $L$  over alphabet  $\Sigma$  is said to be *bounded* if and only if  $L$  is included in (the language of) a bounded expression  $\mathbf{b}$ .

**Theorem 2 ([21]).** *Let  $G = (X, \Sigma, \delta)$  be a grammar, and  $X \in X$  be a nonterminal, such that  $L_X(G)$  is bounded. Then  $L_X(G) = L_X^{(k)}(G)$  for some  $k \geq 1$ .*

The class of programs for which our method is complete is defined below:

**Definition 2.** *Let  $\mathcal{P}$  be a program and  $G_{\mathcal{P}} = (X, \hat{\Theta}, \delta)$  be its corresponding visibly pushdown grammar. Then  $\mathcal{P}$  is said to be *bounded periodic* if and only if:*

1.  $L_X(G_{\mathcal{P}})$  is bounded for each  $X \in X$ ;
2. each relation  $\rho_{\tau}$  occurring in the program, for some  $\tau \in \hat{\Theta}$ , is periodic.

*Example 6.* (continued from Ex. 4) Recall that  $L_{Q_{init}}^{(2)}(G_{\mathcal{P}}) = L_{Q_{init}}^{(2)}(G_{\mathcal{P}})$  which equals to the set  $\{(\tau_1 \tau_2 \langle \tau_3 \rangle^n \tau_1 \tau_5 \tau_6 \tau_7 \langle \tau_3 \rangle \tau_4)^n \mid n \geq 0\} \subseteq (\tau_1 \tau_2 \langle \tau_3 \rangle^* \tau_1^* \tau_5^* \tau_6^* \tau_7^* \langle \tau_3 \rangle \tau_4)^*$ . ■

Concerning condition 1, it is decidable [14] and previous work [16] defined a class of programs following a recursion scheme which ensures boundedness of the set of interprocedurally valid paths. Moreover, when condition 1 does not hold, one can still pick a bounded expression  $\mathbf{b}$  and enforce boundedness by replacing  $G_{\mathcal{P}}$  with grammar  $G'_{\mathcal{P}}$ , such that  $L_X(G'_{\mathcal{P}}) = L_X(G_{\mathcal{P}}) \cap \mathbf{b}$ . Hence  $G'_{\mathcal{P}}$  satisfies condition 1, although at the price of coverage, since interprocedurally valid paths not in  $\mathbf{b}$  have been filtered out.

This section shows that the underapproximation sequence  $\{\llbracket \mathcal{P} \rrbracket^{(k)}\}_{k=1}^{\infty}$ , defined in Section 4, when applied to any bounded periodic programs  $\mathcal{P}$ , always yields  $\llbracket \mathcal{P} \rrbracket$  in finitely many steps, and moreover each iterate  $\llbracket \mathcal{P} \rrbracket^{(k)}$  is computable and Presburger

definable. Furthermore the method can be applied *as it is* to bounded periodic programs, without prior knowledge of the bounded expression  $\mathbf{b} \supseteq L_Q(G_{\mathcal{P}})$ .

The proof goes as follows. Because  $\mathcal{P}$  is bounded periodic, Thm. 2 shows that the semantics  $\llbracket \mathcal{P} \rrbracket$  of  $\mathcal{P}$  coincide with its  $k$ -index semantics  $\llbracket \mathcal{P} \rrbracket^{(k)}$  for some  $k \geq 1$ . Hence, the result of Thm. 1 shows that for each  $q \in n\mathcal{F}(\mathcal{P})$ , the  $k$ -index semantics  $\llbracket \mathcal{P} \rrbracket_q^{(k)}$  is given by the semantics  $\llbracket \mathcal{H} \rrbracket_{query_Q^k}$  of procedure  $query_Q^k$  of the program  $\mathcal{H}$ . Then, because  $\mathcal{P}$  is bounded periodic, we show in Thm. 3 that every procedure  $query_Q^k$  of program  $\mathcal{H}$  is *flattable* (Def. 3). Finally, since all transitions of  $\mathcal{H}$  are periodic and each procedure  $query_Q^k$  is flattable then  $\llbracket \mathcal{P} \rrbracket$  is computable in finite time by existing tools, such as FAST [6] or FLATA [9,8]. In fact, these tools are guaranteed to terminate provided that (a) the input program is flattable; and (b) loops are labeled with periodic relations.

**Definition 3.** Let  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  be a non-recursive program and  $G_{\mathcal{P}} = (X, \hat{\Theta}, \delta)$  be its corresponding visibly pushdown grammar. Procedure  $P_i$  is said to be *flattable* if and only if there exists a bounded and regular language  $R$  over  $\hat{\Theta}$ , such that  $\llbracket \mathcal{P} \rrbracket_{P_i} = \bigcup_{\alpha \in L_{P_i}(G_{\mathcal{P}}) \cap R} \llbracket \alpha \rrbracket$ .

Notice that a flattable program is not necessarily bounded (Def. 2), but its semantics can be computed by looking only at a bounded subset of interprocedurally valid sequence of statements.

**Theorem 3.** Let  $\mathcal{P} = \langle P_1, \dots, P_n \rangle$  be a bounded periodic program, and let  $q \in n\mathcal{F}(\mathcal{P})$ . Then, for any  $k \geq 1$ , procedure  $query_Q^k$  of program  $\mathcal{H}$  is flattable.

The proof of Thm. 3 roughly goes as follows: recall that we have  $\llbracket P \rrbracket_q = \llbracket P \rrbracket_q^{(k)}$  for each  $q \in n\mathcal{F}(\mathcal{P})$  and so it is sufficient to consider the set  $L_Q^{(k)}(G_{\mathcal{P}})$  of interprocedurally valid paths. We further show (Thm. 4) that a strict subset of the  $k$ -index derivations of  $G_{\mathcal{P}}$  is sufficient to capture  $L_Q^{(k)}(G_{\mathcal{P}})$ . Moreover this subset of derivations is characterizable by a bounded expression  $\mathbf{b}_{\Gamma}$  over the productions of  $G_{\mathcal{P}}$ . Then we use  $\mathbf{b}_{\Gamma}$  to give a subset  $f(\mathbf{b}_{\Gamma})$  of the interprocedurally valid path of procedure  $query_Q^k$  of  $\mathcal{H}$  that is sufficient to capture  $\llbracket \mathcal{H} \rrbracket_{query_Q^k}$ . Finally, using existing results, we show (Thm. 5) that  $f(\mathbf{b}_{\Gamma})$  is a bounded and regular set. Hence we conclude that each  $query_Q^k$  is flattable. A full proof of Thm. 3 is given in appendix A.6.

**Control Sets** Given a grammar  $G = (X, \Sigma, \delta)$ , we call any subset of  $\delta^*$  a *control set*. Let  $\Gamma$  be a control set, we denote by  $L_X(\Gamma, G) = \{w \in \Sigma^* \mid \exists \gamma \in \Gamma: X \xrightarrow{\gamma} w\}$ , the set of words resulting from derivations with control word in  $\Gamma$ .

**Depth-first Derivations** are defined as expected:

**Definition 4 ([22]).** Let  $D \equiv X = w_0 \Longrightarrow^* w_n = w$  be a derivation. Let  $k > 0$ ,  $x_i \in \Sigma^*$ ,  $A_i \in X$  such that  $w_m = x_0 A_1 x_1 \cdots A_k x_k$ ; and for each  $i$ ,  $1 \leq i \leq k$ , let  $f_m(i)$  denote the index of the first word in  $D$  in which the particular occurrence of variable  $A_i$  appears. Let  $A_j$  be the nonterminal replaced in step  $w_m \Longrightarrow w_{m+1}$  of  $D$ . Then  $D$  is said to be *depth-first* if and only if for all  $m$ ,  $0 \leq m < n$  we have  $f_m(i) \leq f_m(j)$ , for all  $1 \leq i \leq k$ .

We define the set  $DF_X(G)$  ( $DF_X^{(k)}(G)$ ) of words produced using only depth-first derivations (of index at most  $k$ ) in  $G$  starting from  $X$ . Clearly,  $DF_X(G) \subseteq L_X(G)$  and  $DF_X^{(k)}(G) \subseteq L_X^{(k)}(G)$  for all  $k \geq 1$ . We further define the set  $DF_X(\Gamma, G)$  ( $DF_X^{(k)}(\Gamma, G)$ ) of words produced using depth-first derivations (of index at most  $k$ ) with control words from  $\Gamma$ .

The following theorem shows that  $L_Q^{(k)}(G_P)$  is captured by a subset of depth-first derivations whose control words belong to some bounded expression.

**Theorem 4.** *Let  $G = (X, \Theta, \delta)$  be a visibly pushdown grammar,  $X_0 \in X$  be a nonterminal such that  $L_{X_0}(G)$  is bounded. Then for each  $k \geq 1$  there exists a bounded expression  $\mathbf{b}_\Gamma$  over  $\delta$  such that  $DF_{X_0}^{(k)}(\mathbf{b}_\Gamma, G) = L_{X_0}^{(k)}(G)$ .*

Finally, to conclude that  $query_Q^k$  is flattable, we map the  $k$ -index depth-first derivations of  $G$  into the interprocedurally valid paths of  $query_Q^k$ . Then, applying Thm. 5 on that mapping, we conclude the existence of a bounded and regular set of interprocedurally valid paths of  $query_Q^k$  sufficient to capture its semantics.

**Theorem 5.** *Given two alphabets  $\Sigma$  and  $\Delta$ , let  $f$  be a function from  $\Sigma^*$  into  $\Delta^*$  such that (i) if  $u$  is a prefix of  $v$  then  $f(u)$  is a prefix of  $f(v)$ ; (ii) there exists an integer  $M$  such that  $|f(wa)| - |f(w)| \leq M$  for all  $w \in \Sigma^*$  and  $a \in \Sigma$ ; (iii)  $f(\epsilon) = \epsilon$ ; (iv)  $f^{-1}(R)$  is regular for all regular languages  $R$ . Then  $f$  preserves regular sets. Furthermore, for each bounded expression  $\mathbf{b}$  we have that  $f(\mathbf{b})$  is bounded.*

## 6 Experiments

We have implemented the proposed method in the FLATA verifier [18] and experimented with several benchmarks. First, we have considered several programs, taken from [1], that perform arithmetic and logical operations in a recursive way such as `plus` (addition), `timesTwo` (multiplication by two), `leq` (comparison), and `parity` (parity checking). It is worth noting that these programs have finite index and stabilization of the underapproximation sequence is thus guaranteed. Our technique computes summaries by verifying that  $\llbracket \mathcal{P} \rrbracket^{(2)} = \llbracket \mathcal{P} \rrbracket^{(3)}$  for all these benchmarks, see Table 1 (the platform used for experiments is Intel® Core™2 Duo CPU P8700, 2.53GHz with 4GB of RAM).

$$F_a(x) = \begin{cases} x - 10 & \text{if } x \geq 101 \\ (F_a)^a(x + 10 \cdot a - 9) & \text{if } x \leq 100 \end{cases} \quad G_b(x) = \begin{cases} x - 10 & \text{if } x \geq 101 \\ G(G(x + b)) & \text{if } x \leq 100 \end{cases}$$

Next, we have considered the generalized McCarthy 91 function [10], a well-known verification benchmark that has long been a challenge. We have automatically computed precise summaries of its generalizations  $F_a$  and  $G_b$  above for  $a = 2, \dots, 8$  and  $b = 12, \dots, 14$ . The indices of the recursive programs implementing the  $F_a, G_b$  functions are not bounded, however the sequence reached the fixpoint after at most 4 steps.

Program	Time [s]	$k$
<code>timesTwo</code>	0.7	2
<code>leq</code>	0.7	2
<code>parity</code>	0.8	2
<code>plus</code>	3.4	2
$F_{a=2}$	3.7	3
$F_{a=8}$	45.1	4
$G_{b=12}$	5.7	3
$G_{b=13}$	19.1	3
$G_{b=14}$	24.2	3

**Table 1.** Experiments.

## References

1. Termination Competition 2011. <http://termcomp.uibk.ac.at/termcomp/home.seam>
2. Albarghouthi, A., Gurfinkel, A., Chechik, M.: Whale: An interpolation-based algorithm for inter-procedural verification. In: VMCAI'12. LNCS, vol. 7148, pp. 39–55. Springer (2012)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. JACM 56(3), 16 (2009)
4. Atig, M.F., Ganty, P.: Approximating petri net reachability along context-free traces. In: FSTTCS'11. LIPIcs, vol. 13, pp. 152–163. Schloss Dagstuhl (2011)
5. B. Cook, A.P., Rybalchenko, A.: Summarization for termination: no return! Formal Methods in System Design 35, 369–387 (2009)
6. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Fast acceleration of symbolic transition systems. In: CAV'03. LNCS, vol. 2725, pp. 118–121. Springer (2003)
7. Boigelot, B.: Symbolic Methods for Exploring Infinite State Spaces. Ph.D. thesis, University of Liège (1998)
8. Bozga, M., Iosif, R., Konečný, F.: Fast acceleration of ultimately periodic relations. In: CAV'10. LNCS, vol. 6174, pp. 227–242. Springer (2010)
9. Bozga, M., Iosif, R., Lakhnech, Y.: Flat parametric counter automata. Fundamenta Informaticae 91(2), 275–303 (2009)
10. Cowles, J.: Computer-aided reasoning. chap. Knuth's generalization of McCarthy's 91 function, pp. 283–299 (2000)
11. Esparza, J., Kiefer, S., Luttenberger, M.: Newtonian program analysis. JACM 57(6), 33:1–33:47 (2010)
12. Finkel, A., Leroux, J.: How to compose presburger-accelerations: Applications to broadcast protocols. In: FSTTCS'02. LNCS, vol. 2556, pp. 145–156. Springer (2002)
13. Ganty, P., Majumdar, R., Monmege, B.: Bounded underapproximations. Formal Methods in System Design 40(2), 206–231 (2012)
14. Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill, Inc., New York, NY, USA (1966)
15. Godefroid, P., Nori, A.V., Rajamani, S.K., Tetali, S.: Compositional may-must program analysis: unleashing the power of alternation. In: POPL'10. pp. 43–56. ACM (2010)
16. Godoy, G., Tiwari, A.: Invariant checking for programs with procedure calls. In: SAS'09. LNCS, vol. 5673, pp. 326–342. Springer (2009)
17. Gruska, J.: A few remarks on the index of context-free grammars and languages. Information and Control 19(3), 216–223 (1971)
18. Hojjat, H., Konečný, F., Garnier, F., Iosif, R., Kuncak, V., Rümmer, P.: A verification toolkit for numerical transition systems - tool paper. In: FM. pp. 247–251 (2012)
19. Lalire, G., Argoud, M., Jeannet, B.: Interproc. <http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interp>
20. Latteux, M.: Mots infinis et langages commutatifs. Informatique Théorique et Applications 12(3) (1978)
21. Luker, M.: A family of languages having only finite-index grammars. Information and Control 39(1), 14–18 (1978)
22. Luker, M.: Control sets on grammars using depth-first derivations. Mathematical Systems Theory 13, 349–359 (1980)
23. Reps, T., Horwitz, S., Sagiv, M.: Precise interprocedural dataflow analysis via graph reachability. In: POPL'95. pp. 49–61. ACM (1995)
24. Sharir, M., Pnueli, A.: Two approaches to interprocedural data flow analysis. In: Program Flow Analysis: Theory and Applications, chap. 7, pp. 189–233. Prentice-Hall, Inc. (1981)

## A Missing material

### A.1 Proof of Lemma 1

*Proof:* The proof of Properties 1 and 2 follow immediately from the definition of  $\xRightarrow{(k)}$ . Let us now turn to the proof of Property 3 (only if). First we define  $w_1$  and  $w_2$ . Take the derivation  $BC \xRightarrow{(k)}^* w$  and look at the last step. It must be of the form  $xYz \xRightarrow{(k)} xyz = w$  and one of the following must hold: either  $Y$  has been generated from  $B$  or from  $C$ . Suppose that  $Y$  stems from  $C$  (the other case is treated similarly). In this case, transitively remove from the derivation all the steps transforming the rightmost occurrence of  $C$ . Hence we obtain a derivation  $BC \xRightarrow{(k)}^* w_1C$ . Then  $w_2$  is the unique word satisfying  $w = w_1w_2$ . Since  $BC \xRightarrow{(k)}^* w_1C$ , we find by removing the occurrence of  $C$  in rightmost position at every step that  $B \xRightarrow{(k-1)}^* w_1$  and we are done. Having  $Y$  stemming from  $B$  yields  $C \xRightarrow{(k-1)}^* w_2$ . For the proof of the other direction (if) assuming (i) (the other case is similar), it is easily seen that  $BC \xRightarrow{(k)}^* w_1C \xRightarrow{(k)}^* w_1w_2$ .  $\square$

### A.2 Proof of Lemma 2 only if direction

*Proof:* Recall that in this proof we use the tail-recursive version of Algorithm 1 which is obtained by replacing every two statements of the form  $PC \leftarrow X; \mathbf{goto\ start};$  by  $\mathit{query}_X^k(\mathbf{x}_I, \mathbf{x}_O); \mathbf{return};$  (as it appears in the comments of Alg. 1).

“ $\Rightarrow$ ” Let  $I \cdot O \in \mathbb{Z}^{x \times x}$  such that the call to  $\mathit{query}_Q^k(I, O)$  returns, that is, with parameters  $I$  and  $O$  procedure  $\mathit{query}_Q^k$  has an execution that terminates with an empty call stack. We show that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$  by induction on the number of times  $\ell \geq 1$  a procedure of  $\mathcal{H}$  is invoked. If  $\ell = 1$  then the only invocation is  $\mathit{query}_Q^k(I, O)$ . So it is necessarily the case that, at the non-deterministic jump labelled **start**, the destination has the form  $\mathbf{p}_i^a$  for  $1 \leq i \leq n_a$ . Further, label  $\mathbf{p}_i^a$  corresponds to a production of the form  $Q \rightarrow \tau$  of  $\delta$ , hence we find that  $\tau \in L_Q^{(k)}(G_P)$  since  $k \geq 1$ . Next, because the **assume** statement succeeds, we find that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \rho_\tau$ , hence that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \tau \rrbracket$ , next that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \bigcup_{\alpha \in L_Q^{(k)}(G_P)} \llbracket \alpha \rrbracket$ , and finally that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$  by definition of  $k$ -index semantics and we are done.

If  $\ell > 1$ , there are two possibilities for the first call to a procedure of  $\mathcal{H}$  following the call  $\mathit{query}_Q^k(I, O)$ .

- We are in the case  $\mathit{tail}(p_i^b) = \tau Q'$  for some  $1 \leq i \leq n_b$  and so  $\mathit{query}_Q^k(I, O)$  executes **havoc**( $\mathbf{x}_J$ ), **assume**  $\rho_\tau(\mathbf{x}_I, \mathbf{x}_J)$ ,  $\mathbf{x}_J \leftarrow \mathbf{x}_J$ , followed by  $\mathit{query}_{Q'}^k(\mathbf{x}_I, \mathbf{x}_O)$  then **return**. Lets us denote by  $I$  and  $J$  the content of  $\mathbf{x}_I$  before and after the assignment. By induction hypothesis, we find that  $\langle J \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_{q'}^{(k)}$ , hence that there exists  $\alpha \in L_{Q'}^{(k)}(G_P)$  such that  $\langle J \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \alpha \rrbracket$ . Next  $p_i^b$  corresponds to a production of the form  $Q \rightarrow \tau Q'$  of  $\delta$ , hence we find that  $\tau\alpha \in L_Q^{(k)}(G_P)$  since  $k \geq 1$ . Then, since



$\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle = \langle I \downarrow_{\mathbf{x}_i}, J \downarrow_{\mathbf{x}_i} \rangle \circ \langle J \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \rho_\tau \circ \llbracket \alpha \rrbracket$ , we find that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \tau \alpha \rrbracket$  by definition of the semantics, hence that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \bigcup_{\alpha \in L_Q^{(k)}(G_{\mathcal{P}})} \llbracket \alpha \rrbracket$  and finally

that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$  by definition and we are done.

- We are in the case  $\text{tail}(p_i^c) = \langle \tau Q_j^{\text{init}} \tau \rangle Q'$  for some  $1 \leq i \leq n_c$ . We further assume that the **rod** branch is executed (the **ord** being treated similarly). Therefore  $\text{query}_Q^k(I, O)$  executes **havoc**( $\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$ ), **assume**  $\rho_{\langle \tau \rangle}(\mathbf{x}_I, \mathbf{x}_J)$ , **assume**  $\rho_{\langle \tau \rangle}(\mathbf{x}_K, \mathbf{x}_L)$ , **assume**  $\phi_\tau(\mathbf{x}_I, \mathbf{x}_L)$ ,  $\mathbf{x}_I \leftarrow \mathbf{x}_J$ ,  $\mathbf{x}_O \leftarrow \mathbf{x}_K$ , followed by the calls  $\text{query}_{Q'}^{k-1}(\mathbf{x}_L, \mathbf{x}_O)$ ,  $\text{query}_{Q_j^{\text{init}}}^k(\mathbf{x}_I, \mathbf{x}_O)$  and then **return**. Call  $I, J, K \in \mathbb{Z}^{\mathbf{x}}$  the values picked by **havoc**.

Following the induction hypothesis, we find that  $\langle L \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_{q'}^{(k-1)}$  and  $\langle J \downarrow_{\mathbf{x}_j}, K \downarrow_{\mathbf{x}_j} \rangle \in \llbracket \mathcal{P} \rrbracket_{q_j^{\text{init}}}^{(k)}$ . This implies that there exists  $\alpha \in L_{Q'}^{(k-1)}(G_{\mathcal{P}})$  and  $\alpha' \in L_{Q_j^{\text{init}}}^{(k)}(G_{\mathcal{P}})$  such that  $\langle L \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \alpha \rrbracket$  and  $\langle J \downarrow_{\mathbf{x}_j}, K \downarrow_{\mathbf{x}_j} \rangle \in \llbracket \alpha' \rrbracket$ . Moreover, the definition of  $p_i^c$  and Lem. 1 (prop. 3) shows that  $\langle \tau \alpha' \tau \rangle \alpha \in L_Q^{(k)}(G_{\mathcal{P}})$

Next,  $\langle I \downarrow_{\mathbf{x}_i}, J \downarrow_{\mathbf{x}_j} \rangle \in \rho_{\langle \tau \rangle}$ ,  $\langle J \downarrow_{\mathbf{x}_j}, K \downarrow_{\mathbf{x}_j} \rangle \in \llbracket \alpha' \rrbracket$ ,  $\langle K \downarrow_{\mathbf{x}_j}, L \downarrow_{\mathbf{x}_i} \rangle \in \rho_\tau$  and  $\langle I \downarrow_{\mathbf{x}_i}, L \downarrow_{\mathbf{x}_i} \rangle \in \phi_\tau$  shows that  $\langle I \downarrow_{\mathbf{x}_i}, L \downarrow_{\mathbf{x}_i} \rangle \in R_\tau$  as given in the semantics. Again by the semantics, we find that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \langle \tau \alpha' \tau \rangle \alpha \rrbracket$ , hence that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \bigcup_{\gamma \in L_Q^{(k)}(G_{\mathcal{P}})} \llbracket \gamma \rrbracket$ , and

finally that  $\langle I \downarrow_{\mathbf{x}_i}, O \downarrow_{\mathbf{x}_i} \rangle \in \llbracket \mathcal{P} \rrbracket_q^{(k)}$  by definition of  $k$ -index semantics and we are done.  $\square$

### A.3 Examples of Periodic Relations

An *octagonal relation* is defined by a finite conjunction of constraints of the form  $\pm x \pm y \leq c$ , where  $x$  and  $y$  range over the set  $\mathbf{x} \cup \mathbf{x}'$ , and  $c$  is an integer constant. The transitive closure of any octagonal relation has been shown to be Presburger definable and effectively computable [8].

A *linear affine relation* is defined by a formula  $\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv C\mathbf{x} \geq \mathbf{d} \wedge \mathbf{x}' = A\mathbf{x} + \mathbf{b}$ , where  $A \in \mathbb{Z}^{n \times n}$ ,  $C \in \mathbb{Z}^{p \times n}$  are matrices and  $\mathbf{b} \in \mathbb{Z}^n$ ,  $\mathbf{d} \in \mathbb{Z}^p$ .  $\mathcal{R}$  is said to have the *finite monoid property* if and only if the set  $\{A^i \mid i \geq 0\}$  is finite. It is known that the finite monoid condition is decidable [7], and moreover that the transitive closure of a finite monoid affine relation is Presburger definable and effectively computable [12,7].

### A.4 Proof of Theorem 5

**Definition 5 ([14]).** A generalized sequential machine, *abbreviated gsm*, is a 6-tuple  $S = (K, \Sigma, \Delta, \delta, \lambda, q_1)$  where

- $K$  is a finite nonempty set (of states).
- $\Sigma$  is an alphabet (of inputs).
- $\Delta$  is an alphabet (of outputs).
- $\delta$  (the next-state function) is a mapping of  $K \times \Sigma$  into  $K$ .
- $\lambda$  (the output function) is a mapping of  $K \times \Sigma$  into  $\Delta^*$ .
- $q_1$  is a distinguished element of  $K$  (the start state).

The functions  $\delta$  and  $\lambda$  are extended by induction to  $K \times \Sigma^*$  by defining for every state  $q$ , every word  $x \in \Sigma^*$ , and every  $y$  in  $\Sigma$

- $\delta(q, \varepsilon) = q$  and  $\lambda(q, \varepsilon) = \varepsilon$ .
- $\delta(q, xy) = \delta[\delta(q, x), y]$  and  $\lambda(q, xy) = \lambda(q, x)\lambda[\delta(q, x), y]$ .

It is readily seen that the second item holds for all words  $x$  and  $y$  in  $\Sigma^*$ .

**Definition 6 ([14]).** Let  $S = (K, \Sigma, \Delta, \delta, \lambda, q_1)$  be a gsm. The operation defined by  $S(x) = \lambda(q_1, x)$  for each  $x \in \Sigma^*$  is called a gsm mapping.

**Theorem 6 (Theorem 3.4.1 (if direction), [14]).** Let  $f$  be a function from  $\Sigma^*$  into  $\Delta^*$  such that (i)  $f$  preserves prefixes, that is if  $u$  is a prefix of  $v$  then  $f(u)$  is a prefix of  $f(v)$ ; (ii)  $f$  has bounded outputs, that is, there exists an integer  $M$  such that  $|f(wa)| - |f(w)| \leq M$  for all  $w \in \Sigma^*$  and  $a \in \Sigma$ ; (iii)  $f(\varepsilon) = \varepsilon$ ; (iv)  $f^{-1}(R)$  is regular for all regular languages  $R$ . Then  $f$  is a gsm mapping.

**Theorem 7 (Theorem 3.3.2, [14]).** Each gsm mapping preserves regular sets.

**Lemma 3 (Lemma 5.5.3, [14]).**  $S(w_1^* \dots w_n^*)$  is bounded for each gsm  $S$  and all words  $w_1, \dots, w_n$ .

Finally, Theorem 5 is an easy consequence of the above facts.

## A.5 Proof of Theorem 4

The proof is long but technically not difficult. First, we need to introduce some new material. The *Szilar language* of a grammar  $G = (X, \Sigma, \delta)$  and denoted  $Sz_X(G) \subseteq \delta^*$  is the set of control words used in the derivations of  $G$  starting with  $X \in X$ . We denote by  $Sz_X^{df}(G)$  the set of control words used in the depth first derivations of  $G$  starting with  $X$ . Moreover let  $Sz_X^{df}(G, k)$  denote the set of control words used in depth first  $k$ -index derivations of  $G$  starting with  $X$ . Next, we recall a couple of known results [22, 20].

**Lemma 4 ([22]).** For all  $k \geq 1$ , we have  $DF_X^{(k)}(G) = L_X^{(k)}(G)$  and  $Sz_X^{df}(G, k)$  is regular.

Given an alphabet  $\Sigma = \{u_1, \dots, u_k\}$ , let  $Pk(u_i) = \mathbf{e}_i$  be the  $k$ -dimensional vector having 1 on the  $i$ -th position and 0 everywhere else. We define  $Pk(\varepsilon) = \mathbf{0}$ ,  $Pk(u_{i_1} \dots u_{i_n}) = \sum_{j=1}^n Pk(u_{i_j})$  for any word  $u_{i_1} \dots u_{i_n} \in \Sigma^*$  and  $Pk(L) = \{Pk(w) \mid w \in L\}$  for any language  $L \subseteq \Sigma^*$ . The following result was proved in [13]:

**Theorem 8 (Thm. 1 from [13], also in [20]).** For every regular language  $L$  there exists a bounded expression  $\mathbf{b}_\Gamma$  such that  $Pk(L \cap \mathbf{b}_\Gamma) = Pk(L)$ .

Next we prove a result characterizing a subset of derivations sufficient to capture a bounded context-free language.

**Lemma 5.** Let  $G = (X, \Sigma, \delta)$  be a grammar and  $X \in X$  be a nonterminal, such that  $L_X(G) \subseteq a_1^* \dots a_d^*$  where  $a_1, \dots, a_d$  are distinct symbols of  $\Sigma$ . Then, for each  $k \geq 1$  there exists a bounded expression  $\mathbf{b}_\Gamma$  over  $\delta$  such that  $DF_X^{(k)}(\mathbf{b}_\Gamma, G) = L_X^{(k)}(G)$ .

*Proof:* We first establish the claim that for each  $k \geq 1$ , there exists a bounded expression  $\mathbf{b}_\Gamma$  over  $\delta$  such that  $Pk(Sz_X^{df}(G, k) \cap \mathbf{b}_\Gamma) = Pk(Sz_X^{df}(G, k))$ . By Lemma 4,  $Sz_X^{df}(G, k)$  is a regular language, and by Theorem 8, there exists a bounded expression  $\mathbf{b}_\Gamma$  over  $\delta$  such that  $Pk(Sz_X^{df}(G, k) \cap \mathbf{b}_\Gamma) = Pk(Sz_X^{df}(G, k))$  which proves the claim. Next we prove that  $DF_X^{(k)}(\mathbf{b}_\Gamma, G) = L_X^{(k)}(G)$ .

Let  $\delta = \langle X_i \rightarrow v_i \rangle_{i=1}^m$  be the sequence of productions of  $G$ , taken in some fixed order. For each right-hand side  $v_i$  of a production in  $\delta$ , let  $pk(v_i) \in \mathbb{Z}^d$  be the Parikh image of the subword of obtained by taking the projection of  $v_i$  on  $a_1, \dots, a_d$ . Let  $\Pi = [pk(v_i)]_{i=1}^m$  be the  $m \times d$  matrix whose rows are the  $pk(v_i)$  vectors. Let  $X \xrightarrow{\gamma} w$ . Then we have  $Pk(w) = Pk(\gamma) \times \Pi$ , and consequently,  $Pk(\gamma_1) = Pk(\gamma_2)$  implies that  $Pk(w_1) = Pk(w_2)$  for any two derivations  $X \xrightarrow{\gamma_i} w_i$  of  $G$ ,  $i = 1, 2$ . Moreover, the assumption  $L_X(G) \subseteq a_1^* \dots a_d^*$  where  $a_1, \dots, a_d$  are distinct symbols shows that we further have  $w_1 = w_2$ .

We prove that  $L_X^{(k)}(G) \subseteq DF_X^{(k)}(\mathbf{b}_\Gamma, G)$ , the other direction being immediate. By Lemma 4, we have  $L_X^{(k)}(G) = DF_X^{(k)}(G)$ . Let  $w \in DF_X^{(k)}(G)$  be a word, and  $X \xrightarrow{\gamma} w$  be a depth-first derivation of  $w$ . Since  $Pk(Sz_X^{df}(G, k) \cap \mathbf{b}_\Gamma) = Pk(Sz_X^{df}(G, k))$ , there exists a control word  $\beta \in Sz_X^{df}(G, k) \cap \mathbf{b}_\Gamma$  such that  $Pk(\beta) = Pk(\gamma)$ , hence  $X \xrightarrow{\beta} w'$  and  $w' = w$  as shown above.  $\square$

For the rest of this section, let  $G = (\mathcal{X}, \Theta, \delta)$  be a visibly pushdown grammar (we ignore for the time being the distinction between tagged and untagged alphabet symbols), and  $X_0 \in \mathcal{X}$  be an arbitrarily chosen nonterminal, and let  $\mathbf{b} = w_1^* \dots w_d^*$  be a bounded expression, where  $w_i = b_1^{(i)} \dots b_{j_i}^{(i)} \in \Theta^*$ , for every  $1 \leq i \leq d$ . Let  $G^{\mathbf{b}} = (\mathcal{X}^{\mathbf{b}}, \Theta, \delta^{\mathbf{b}})$  be the regular grammar, where  $\mathcal{X}^{\mathbf{b}} = \{q_r^{(s)} \mid 1 \leq s \leq d \wedge 1 \leq r \leq j_s\}$  and:

$$\delta^{\mathbf{b}} = \left\{ q_i^{(s)} \rightarrow b_i^{(s)} q_{i+1}^{(s)} \mid 1 \leq s \leq d \wedge 1 \leq i < j_s \right\} \quad (1)$$

$$\cup \left\{ q_{j_s}^{(s)} \rightarrow b_{j_s}^{(s)} q_1^{(s')} \mid 1 \leq s \leq s' \leq d \right\} \quad (2)$$

$$\cup \left\{ q_1^{(s)} \rightarrow \varepsilon \mid 1 \leq s \leq n \right\}.$$

It is routine to check that  $\bigcup_{s=1}^d L_{q_1^{(s)}}(G^{\mathbf{b}}) = w_1^* \dots w_d^*$ . Next, we define  $G^{\boxtimes} = (\mathcal{X}^{\boxtimes}, \Theta, \delta^{\boxtimes})$ :

- $\mathcal{X}^{\boxtimes} = \{X_0^{\boxtimes}\} \cup \{[q_r^{(s)} X q_y^{(x)}] \mid X \in \mathcal{X} \wedge q_r^{(s)} \in \mathcal{X}^{\mathbf{b}} \wedge q_y^{(x)} \in \mathcal{X}^{\mathbf{b}} \wedge s \leq x\}$
- $\delta^{\boxtimes}$  contains, for every  $1 \leq s \leq x \leq n$ , a production  $X_0^{\boxtimes} \rightarrow [q_1^{(s)} X_0 q_1^{(x)}]$ , and:
  - for every production  $X \rightarrow \tau \in \delta$

$$[q_r^{(s)} X q_y^{(x)}] \rightarrow \tau \in \delta^{\boxtimes} \quad \text{if } q_r^{(s)} \rightarrow \tau q_y^{(x)} \in \delta^{\mathbf{b}} \quad (3)$$

- for every production  $X \rightarrow \tau Y \in \delta$

$$[q_r^{(s)} X q_y^{(x)}] \rightarrow \tau [q_t^{(z)} Y q_z^{(x)}] \in \delta^{\boxtimes} \quad \text{if } q_r^{(s)} \rightarrow \tau q_t^{(z)} \in \delta^{\mathbf{b}} \quad (4)$$

- for every production  $X \rightarrow \tau Z \sigma Y \in \delta$

$$\begin{aligned} [q_r^{(s)} X q_y^{(x)}] &\rightarrow \tau [q_t^{(z)} Y q_u^{(v)}] \sigma [q_k^{(\ell)} Z q_y^{(x)}] \\ &\text{if } q_r^{(s)} \rightarrow \tau q_t^{(z)} \in \delta^{\mathbf{b}} \text{ and } q_u^{(v)} \rightarrow \sigma q_k^{(\ell)} \in \delta^{\mathbf{b}} \end{aligned} \quad (5)$$

The set  $\delta^{\boxtimes}$  contains no other productions. For each nonterminal  $[q_r^{(s)} X q_y^{(x)}] \in X^{\boxtimes}$ , we define  $\xi([q_r^{(s)} X q_y^{(x)}]) = X$ . Further  $\xi(X_0^{\boxtimes}) = X_0$ . This notation is extended to productions from  $\delta^{\boxtimes}$ , hence sequences of productions in the obvious way. Further we define  $\xi(\Gamma) = \{\xi(d) \mid d \in \Gamma\}$  where  $\Gamma$  is a control set over  $\delta^{\boxtimes}$ . Finally, for a derivation  $D^{\boxtimes} \equiv X_0^{\boxtimes} \Rightarrow [q_1^{(s)} X_0 q_1^{(x)}] \xRightarrow{*} w$  in  $G^{\boxtimes}$ , let  $\xi(D^{\boxtimes}) \equiv X_0 \xRightarrow{*} w$  be the derivation of  $G$  obtained by applying  $\xi$  to each production  $p$  in  $D^{\boxtimes}$ .

**Lemma 6.** *Let  $G = (X, \Theta, \delta)$  be a visibly pushdown grammar,  $X_0 \in X$  be a nonterminal such that  $L_{X_0}(G) \subseteq \mathbf{b}$  for a bounded expression  $\mathbf{b} = w_1^* \dots w_d^*$ . Then for every  $k \geq 1$ , the following hold:*

1.  $L_{X_0}^{(k)}(G) = L_{X_0^{\boxtimes}}^{(k)}(G^{\boxtimes})$
2. Given a control set  $\Gamma$  over  $\delta^{\boxtimes}$  such that  $DF_{X_0^{\boxtimes}}^{(k)}(\Gamma, G^{\boxtimes}) = L_{X_0^{\boxtimes}}^{(k)}(G^{\boxtimes})$  then the control set  $\Gamma' = \xi(\Gamma)$  over  $\delta$  satisfies  $DF_{X_0}^{(k)}(\Gamma', G) = L_{X_0}^{(k)}(G)$ .

*Proof.* (sketch) The proof of point 1 is by induction. So we actually show the following stronger statement. Let  $k \geq 1$  and let  $w \in \Sigma^*$ . We show that  $[q_r^{(s)} X q_v^{(u)}] \xRightarrow[G^{\boxtimes}]^{(k)} w$  iff  $q_r^{(s)} \xRightarrow{*} w q_v^{(u)}$  and  $X \xRightarrow[G]^{(k)} w$ . The proof for the if direction is by induction on the length of  $X \xRightarrow[G]^{(k)} w$ .

“ $i = 1$ ” Then  $X \xRightarrow[G]^{(k)} w$  for some production  $X \rightarrow \tau$  of  $\delta$  with  $w = \tau$ . Also  $q_r^{(s)} \rightarrow \tau q_v^{(u)}$  in  $\delta^{\mathbf{b}}$  and so by definition of  $G^{\boxtimes}$  we have  $[q_r^{(s)} X q_v^{(u)}] \rightarrow \tau$  in  $\delta^{\boxtimes}$  and we are done.

“ $i > 1$ ” We do a case analysis according to the tail of the first production in  $X \xRightarrow[G]^{(k)} w$ .

- $X \xRightarrow[G]^{(k)} \tau X' \xRightarrow[G]^{(k)} \tau w' = w$  which implies that  $X \rightarrow \tau X'$  is in  $\delta$ . Further,  $q_r^{(s)} \xRightarrow{*} w q_v^{(u)}$  shows that there exists  $q_r^{(s)} \Rightarrow \tau q_{r'}^{(s')} \Rightarrow \tau w' q_v^{(u)}$ , hence that  $q_r^{(s)} \rightarrow \tau q_{r'}^{(s')}$  is in  $\delta^{\mathbf{b}}$ , and finally find that  $[q_r^{(s)} X q_v^{(u)}] \rightarrow \tau [q_{r'}^{(s')} X' q_v^{(u)}]$  belongs to  $\delta^{\boxtimes}$ . Also we conclude from the hypothesis that  $X' \xRightarrow[G]^{(k)} w'$  and  $q_{r'}^{(s')} \xRightarrow{*} w' q_v^{(u)}$  and so, by induction hypothesis, we find that  $[q_{r'}^{(s')} X' q_v^{(u)}] \xRightarrow[G^{\boxtimes}]^{(k)} w'$  and we are done.

- $X \xRightarrow[G]^{(k)} \tau X_1 \sigma X_2 \xRightarrow[G]^{(k)} \tau w_1 \sigma w_2 = w$  and so there exists  $X \rightarrow \tau X_1 \sigma X_2$  in  $\delta$ . Moreover, since  $q_s^{(r)} \xRightarrow{*} w q_v^{(u)}$  we find that there exist  $q_s^{(r)} \Rightarrow \tau q_a^{(b)} \Rightarrow \tau w_1 q_{a'}^{(b')} \Rightarrow$

$\tau w_1 \sigma q_c^{(d)} \Rightarrow^* \tau w_1 \sigma w_2 q_v^{(u)}$ . Hence, the definition of  $G^{\bowtie}$  shows that  $[q_s^{(r)} X q_v^{(u)}] \rightarrow \tau[q_a^{(b)} X_1 q_{a'}^{(b')}] \sigma[q_c^{(d)} X_2 q_v^{(u)}]$ . On the other hand, since  $X_1 X_2 \xRightarrow[G]{(k)}^* w_1 w_2$  (simply delete  $\tau$  and  $\sigma$ ), Lemma 1 shows that either  $X_1 \xRightarrow[G]{(k-1)}^* w_1$  and  $X_2 \xRightarrow[G]{(k)}^* w_2$ ; or  $X_1 \xRightarrow[G]{(k)}^* w_1$  and  $X_2 \xRightarrow[G]{(k-1)}^* w_2$ . Let us assume the latter holds (the other being treated similarly). Applying the induction hypothesis, we find that  $[q_a^{(b)} X_1 q_{a'}^{(b')}] \xRightarrow[G^{\bowtie}]{(k)}^* w_1$  and  $[q_c^{(d)} X_2 q_v^{(u)}] \xRightarrow[G^{\bowtie}]{(k-1)}^* w_2$ , hence we conclude the case with the  $k$ -index derivation  $[q_s^{(r)} X q_v^{(u)}] \xRightarrow[G^{\bowtie}]{(k)}^* \tau[q_a^{(b)} X_1 q_{a'}^{(b')}] \sigma[q_c^{(d)} X_2 q_v^{(u)}] \xRightarrow[G^{\bowtie}]{(k)}^* \tau[q_a^{(b)} X_1 q_{a'}^{(b')}] \sigma w_2 \xRightarrow[G]{(k)}^* \tau w_1 \sigma w_2$ .

To conclude the “if” case, observe that  $L_{X_0}^{(k)}(G) \subseteq w_1^* \dots w_d^*$  implies that for every  $w \in L_{X_0}^{(k)}(G)$  we have  $X_0 \xRightarrow[G]{(k)}^* w$  and also  $q_1^{(s)} \Rightarrow^* w q_1^{(s')}$ , hence that  $w \in L_{X_0^{\bowtie}}^{(k)}(G)$ .

Using a similar induction on the length of derivation  $[q_r^{(s)} X q_v^{(u)}] \xRightarrow[G^{\bowtie}]{(k)}^* w$ , the “only if” direction is easily proved.

For the proof of point 2. the “ $\subseteq$ ” is obvious by definition of depth-first derivation. For the reverse direction “ $\supseteq$ ” point 1 shows that  $L_{X_0}^{(k)}(G) = L_{X_0^{\bowtie}}^{(k)}(G^{\bowtie})$ , hence using the assumption we find that  $DF_{X_0^{\bowtie}}^{(k)}(\Gamma, G^{\bowtie}) = L_{X_0}^{(k)}(G)$ . So let  $D \equiv X_0^{\bowtie} \xRightarrow[G^{\bowtie}]{(k)}^* w$  be a depth-first  $k$ -index derivation of  $G^{\bowtie}$  with control word conforming to  $\Gamma$ . Now consider  $\xi(D)$ , it defines again a depth-first  $k$ -index derivation. Further, the definition of  $\xi$  shows that the word generated by  $\xi(D)$  is  $w$ .  $\square$

Let  $\mathcal{A} = \{a_1, \dots, a_d\}$  be an alphabet disjoint from  $\Theta$ , and a language homomorphism  $h: \mathcal{A} \rightarrow \Theta^*$ , defined as  $h(a_i) = w_i$ , for all  $1 \leq i \leq d$ . We now obtain from  $G^{\bowtie}$  a grammar  $G^a$ , over  $\mathcal{A}$ , such that  $L_{X_0}(G^{\bowtie}) = h(L_{X_0}(G^a))$ . Define  $G^a = (X^{\bowtie}, \mathcal{A}, \delta^a)$ , as the result of applying onto  $G^{\bowtie}$  the following transformation on every  $p \in \delta^{\bowtie}$ : if  $p$  was defined using a production  $q_r^{(s)} \rightarrow \gamma q_x^{(y)} \in \delta^p$  where  $r = j_s$  then replace the corresponding occurrence of  $\gamma$  in  $p$  by  $a_s$ , else ( $r \neq j_s$ ) replace the corresponding occurrence of  $\gamma$  by  $\varepsilon$ . In this way we can map the productions of  $G^{\bowtie}$  onto productions of  $G^a$ . This mapping is extended to the derivations of  $G^{\bowtie}$ . The information kept within the nonterminal of  $X^{\bowtie}$  is sufficient to also define the reverse mapping, from the productions (derivations) of  $G^a$  back to the productions (derivations) of  $G^{\bowtie}$ . We define the mapping  $v: \delta^a \rightarrow \delta^{\bowtie}$  as follows, for  $a, b \in \mathcal{A} \cup \{\varepsilon\}$ :

$$\begin{aligned} - v([q_r^{(s)} X q_y^{(x)}] \rightarrow a) &= [q_r^{(s)} X q_y^{(x)}] \rightarrow b_r^{(s)} \\ - v([q_r^{(s)} X q_y^{(x)}] \rightarrow a [q_t^{(z)} Y q_y^{(x)}]) &= [q_r^{(s)} X q_y^{(x)}] \rightarrow b_r^{(s)} [q_t^{(z)} Y q_y^{(x)}] \\ - v([q_r^{(s)} X q_y^{(x)}] \rightarrow a [q_t^{(z)} Y q_u^{(v)}] b [q_k^{(\ell)} Z q_y^{(x)}]) &= [q_r^{(s)} X q_y^{(x)}] \rightarrow b_r^{(s)} [q_t^{(z)} Y q_u^{(v)}] b_u^{(v)} [q_k^{(\ell)} Z q_y^{(x)}] \end{aligned}$$

**Lemma 7.** *Let  $G = (X, \Theta, \delta)$  be a visibly pushdown grammar,  $X_0 \in X$  be a nonterminal, and  $w_1^* \dots w_d^*$  be a bounded expression over  $\Theta$ . Also let  $\mathcal{A} = \{a_1, \dots, a_d\}$  be an alphabet*

disjoint from  $\Theta$ , and  $h: \mathcal{A} \rightarrow \Theta^*$  be the homomorphism defined as  $h(a_i) = w_i$ , for all  $1 \leq i \leq d$ . Then for every  $k \geq 1$ , the following hold:

1.  $L_{X_0^\bowtie}^{(k)}(G^a) = h^{-1}(L_{X_0^\bowtie}^{(k)}(G^\bowtie)) \cap a_1^* \cdots a_d^*$
2. given a control set  $\Gamma^a$  over  $\delta^a$  such that  $DF_{X_0^\bowtie}^{(k)}(\Gamma^a, G^a) = L_{X_0^\bowtie}^{(k)}(G^a)$ , then the control set  $\Gamma' = v(\Gamma^a)$  over  $\delta^\bowtie$  satisfies  $DF_{X_0^\bowtie}^{(k)}(\Gamma', G^\bowtie) = L_{X_0^\bowtie}^{(k)}(G^\bowtie)$

*Proof:* (sketch) The proof of Point 1 is by induction showing the following stronger statement: let  $w \in \Theta^*$  then we have  $[q_r^{(s)} X q_v^{(u)}] \xrightarrow[G^\bowtie]{(k)}^* w$  iff  $[q_r^{(s)} X q_v^{(u)}] \xrightarrow[G^a]{(k)}^* h^{-1}(w) \cap a_1^* \cdots a_d^*$ . The proof is done by induction on the length of the derivations similarly to Lemma 6. It follows that  $L_{X_0^\bowtie}^{(k)}(G^a) = \{a_1^{i_1} \cdots a_d^{i_d} \mid w_1^{i_1} \cdots w_d^{i_d} \in L_{X_0^\bowtie}^{(k)}(G^\bowtie)\}$ , hence that  $h(L_{X_0^\bowtie}^{(k)}(G^a)) = L_{X_0^\bowtie}^{(k)}(G^\bowtie)$  by definition of  $h$  and since  $L_{X_0^\bowtie}^{(k)}(G^\bowtie) \subseteq w_1^* \cdots w_d^*$ . For point 2, applying  $h$  on both side of the assumption to obtain  $h(DF_{X_0^\bowtie}^{(k)}(\Gamma^a, G^a)) = h(L_{X_0^\bowtie}^{(k)}(G^a))$ , hence  $h(DF_{X_0^\bowtie}^{(k)}(\Gamma^a, G^a)) = L_{X_0^\bowtie}^{(k)}(G^\bowtie)$  by point 1. To conclude the proof, it is sufficient to show that  $h(DF_{X_0^\bowtie}^{(k)}(\Gamma^a, G^a)) = DF_{X_0^\bowtie}^{(k)}(v(\Gamma^a), G^\bowtie)$ . Again an induction proof is called for.  $\square$

Before proving Theorem 4 we recall the following result about homomorphisms and bounded languages. Let  $g: \Sigma \rightarrow \mathcal{A}^*$  be a homomorphism that maps each symbol of  $\Sigma$  into a word over  $\mathcal{A}$ , and  $L \subseteq w_1^* \cdots w_d^*$  where  $w_1^* \cdots w_d^*$  is a bounded expression. Then  $g(L)$  is also bounded.<sup>9</sup>

Finally the actual proof the Theorem 4 goes as follows.

*Proof:* Let  $\mathcal{A} = \{a_1, \dots, a_d\}$  be an alphabet disjoint from  $\Theta$ , and let  $h: \mathcal{A} \rightarrow \Theta^*$  be the language homomorphism defined by  $h(a_i) = w_i$ , for all  $1 \leq i \leq d$ . By applying Lemma 6 (first point), and then Lemma 7 (first point) we find that  $L_{X_0}^{(k)}(G) = h(L_{X_0^\bowtie}^{(k)}(G^a))$ . Next, applying Lemma 5 on  $L_{X_0^\bowtie}^{(k)}(G^a)$  we obtain a bounded expression  $\mathbf{b}_{\Gamma^a}$  over  $\delta^a$  such that  $DF_{X_0^\bowtie}^{(k)}(\mathbf{b}_{\Gamma^a}, G^a) = L_{X_0^\bowtie}^{(k)}(G^a)$ . Our next step is to apply the results of Lemma 7 (second point), and Lemma 6 (second point) in that order to obtain that  $L_{X_0}^{(k)}(G) = DF_{X_0}^{(k)}(\xi(v(\mathbf{b}_{\Gamma^a})), G)$ . Finally, since  $\mathbf{b}_{\Gamma^a}$  is a bounded expression, and  $\xi$  and  $v$  are homomorphisms (and so is the composition  $\xi \circ v$ ) we have that  $\xi(v(\mathbf{b}_{\Gamma^a}))$  is bounded, hence included in a bounded expression and we are done.  $\square$

## A.6 Proof of Theorem 3

**Lemma 8.** Let  $G = (X, \Theta, \delta)$  be a visibly pushdown grammar such that for all productions  $p \in \delta$  all nonterminals occurring in  $\text{tail}(p)$  are distinct. Let  $X \in X$  and  $\gamma \in \delta^*$ , then there exists at most one depth-first derivation of  $G$  with control word  $\gamma$ , hence at most one word resulting from it.

<sup>9</sup> Alternatively, it can also be shown using Theorem 5.

*Proof:* By contradiction, suppose there exist two depth-first derivations from  $X$  with control word  $p_1 \dots p_n$ . This means that there exists a  $i$ ,  $1 \leq i \leq n$  such that  $X = w_0 \xRightarrow{p_1} w_1 \dots w_{i-1} \xRightarrow{p_i} w_i$  and  $w_i$  contains two occurrences of the nonterminal  $head(p_i)$ , that is  $w_i = \alpha A_1 \beta A_2 \gamma$  where  $A_1 = A_2 = head(p_i)$  and  $\alpha, \beta, \gamma \in (\Sigma \cup \Theta^*)$ . Two cases arises:

1.  $A_1$  and  $A_2$  result from the occurrence of some  $p_j$  with  $j < i$  which contradicts that all nonterminals occurring in  $tail(p_i)$  are distinct.
2.  $A_1$  and  $A_2$  result from the occurrence of  $p_k$  and  $p_l$  with  $k \neq l$  respectively. Following the definition of depth-first derivation  $p_i$  must be applied to  $A_1$  if  $k > l$ ; and to  $A_2$  if  $k < l$ . In either case  $p_i$  can be applied to only one of the two occurrences which contradicts the existence of two depth-first derivations.

□

Note that because the grammars of this paper stems from programs we can then assume without loss of generality that the condition on  $tail(p)$  for every production  $p$  holds for every grammar in this paper.

Finally the proof the Thm. 3 goes as follows:

*Proof:* (sketch) Since  $\mathcal{P}$  is bounded periodic we can apply Theorem 4 showing there exists a bounded expression  $\mathbf{b}_\Gamma$  over  $\delta$  such that  $DF_Q^{(k)}(\mathbf{b}_\Gamma, G_{\mathcal{P}}) = L_Q^{(k)}(G_{\mathcal{P}})$ . Hence we find that  $\llbracket \mathcal{P} \rrbracket_q^{(k)} = \bigcup_{\alpha \in L_Q^{(k)}(G_{\mathcal{P}})} \llbracket \alpha \rrbracket = \bigcup_{\alpha \in DF_Q^{(k)}(\mathbf{b}_\Gamma, G_{\mathcal{P}})} \llbracket \alpha \rrbracket$ .

Let  $\alpha \in DF_Q^{(k)}(\mathbf{b}_\Gamma, G_{\mathcal{P}})$  and let  $\gamma$  be the control word of the derivation  $D_\gamma$  thereof which is unique by Lemma 8. We then prove that  $D_\gamma$  corresponds to a unique inter-procedurally valid path  $\beta$  of  $query_Q^k$ , that is  $\beta \in L_{query_Q^k}(G_{\mathcal{H}})$ . This is however easily seen looking at the code of  $query_Q^k$  whose control flow follows precisely a depth-first  $k$ -index derivation. Because  $\gamma$  the control word over  $\delta$  determines uniquely  $D_\gamma$  hence  $\beta$  we conclude that there exists a function  $f$  that associates each word over  $\hat{\Theta}$  a unique word  $\hat{\Delta}$  (call  $\hat{\Delta}$  the alphabet of  $G_{\mathcal{H}}$ ). Moreover define  $f(\epsilon) = \epsilon$ . Basically,  $f$  maps each production  $p$  of  $D_\gamma$  to a labelled statement  $\mathbf{p}$  in  $\mathcal{H}$ . Moreover between two consecutive labelled statements  $\mathbf{p}$  and  $\mathbf{p}'$   $f$  is stuffing a sequence of statements of  $\mathcal{H}$  which is unique for the reason that  $D_\gamma$  is unique.

Next, we show that  $f(\mathbf{b}_\Gamma)$  is a bounded regular set over  $\hat{\Delta}$  using Thm. 5. To this end, we need to show that  $f$  satisfies the properties (i) to (iv) in Thm. 5. Following the previous explanations on  $f$  that is stuffing sequences of statements between consecutive productions it is seen that (i) holds, also (ii) holds because the number of statements added between any two consecutive productions is bounded, (iii) holds by definition and finally (iv) holds because  $f^{-1}$  consists in deleting statements not referring to productions which clearly preserves regularity.

We then conclude from Thm. 5, that  $f(\mathbf{b}_\Gamma)$  is a bounded and regular language. Back to  $\llbracket \mathcal{H} \rrbracket_{query_Q^k}$ , we find that

$$\llbracket \mathcal{H} \rrbracket_{query_Q^k} = \bigcup_{\gamma \in L_{query_Q^k}} \llbracket \gamma \rrbracket = \bigcup_{\gamma \in L_{query_Q^k} \cap f(\mathbf{b}_\Gamma)} \llbracket \gamma \rrbracket$$

and that  $\llbracket \mathcal{H} \rrbracket_{query_Q^k}$  is flattable since  $f(\mathbf{b}_\Gamma)$  is a bounded regular set.

□